Treball de Fi de Grau

Enginyeria en Tecnologies Industrials

Disseny d'una GAN com a model predictiu del comportament d'un valor borsari en alta freqüència

MEMÒRIA

Autor: Joan Vendrell Gallart

Director: Cecilio Angulo Bahón

Data: Juny 2019

Escola Tècnica Superior

d'Enginyeria Industrial de Barcelona

Resum

L'objectiu d'aquest treball de fi de grau és estudiar les xarxes neuronals generatives antagòniques, més conegudes com a *Generative Adversarial Neural Networks* o GAN, i desenvolupar un model predictiu del comportament temporal d'un valor borsari en alta freqüència. El projecte parteix d'un model de GAN plantejat en un article de recerca [36] i, a partir d'aquest, es desenvolupen un seguit d'implementacions per buscar uns millors resultats. Concretament, es treballarà amb el valor de les accions d'Apple mostrejades en una freqüència de nanosegons. Pel que fa al model, en aquest projecte es planteja l'ús d'una xarxa recurrent *Long Short-Term Memory* (LSTM) com a generador i, per tant, predictor i d'una xarxa convolucional (CNN, per *Convolutional Neural Network*) en una dimensió com a discriminant per tal de poder realitzar l'entrenament antagònic o discriminador. Tots els models s'han implementat mitjançant la llibreria Pytorch en llenguatge Python sobre la plataforma al núvol Google Colab.

Així doncs, el treball es composa d'uns primers capítols explicatius on es desenvolupa la teoria necessària per implementar el model i, posteriorment, uns capítols de desenvolupament del model i anàlisi dels resultats.

Sumari

Índex de figures

Índex de taules

Capítol 1

Introducció

Predir el valor d'un producte borsari és un objectiu important en el món financer. Una predicció precisa significa importants beneficis i una reducció en el risc de les inversions. És per això, que grans companyies financeres aposten per la recerca en tota mena de noves metodologies en la cerca d'una tècnica infal·lible. Una espècie de pedra filosofal que ho converteixi tot en or, en beneficis. Des del creixement d'Internet i de les noves tecnologies, la rapidesa en la inversió ha crescut exponencialment fins a arribar a assolir els nanosegons. Aquest tipus d'inversions és el que es coneix com a *High Frequency Trading* (HFT) i només està a l'abast de grans companyies amb potents ordinadors i robustos algorismes matemàtics. La popularitat d'aquestes tècniques ha crescut a causa dels avantatges que suposa realitzar més operacions en menor temps. Però això ha incrementat la complexitat dels mercats.

Tot i que l'ús de tècniques matemàtiques i la inversió algorítmica en HFT ha crescut amb el temps, el cert és que no és pas senzill predir el comportament dels mercats financers. La complexitat, la dinàmica caòtica i la no estacionalitat de certes variables involucrades, fa molt difícil l'obtenció d'una tècnica eficaç. Diversos investigadors en àrees i temes molt diferents han intentat analitzar dades històriques per buscar patrons en el comportament i han implementat tècniques complexes fonamentades en l'estadística.

Per altra banda, les xarxes generatives antagòniques són un model de xarxa neuronal molt nova presentada per Ian Goodfellow [9] que consisteix en l'entrenament conjunt de dues xarxes neuronals. Una que genera dades similars a una distribució i una altra que intenta discriminar si les dades són reals o generades. La capacitat d'aquesta estructura ha sorprès al món investigador i, actualment, són un important tòpic de recerca. Era, per tant, qüestió de temps que algú intentés implementar una GAN en la problemàtica dels mercats borsaris en HFT. A data d'inici d'aquest projecte, no s'ha trobat cap article que solucionés un problema real d'alta freqüència amb GANs. L'estudi que més s'aproxima al comentat és [36], un article que implementa una GAN per predir el comportament d'un valor en minuts. Tampoc s'ha trobat cap altre article que treballi amb GANs i dades borsàries. En aquest projecte es partirà de [36] i s'afrontarà un problema real d'HFT.

1.1 Motivació

La principal motivació d'aquest projecte és l'estudi de les xarxes neuronals. Les xarxes neuronals són una tècnica que recentment està demostrant el seu potencial en diferents camps. És per això que, des de fa poc, s'estan començant a implementar xarxes neuronals per solucionar problemes en àmbits molt diversos, des de la medicina a l'anàlisi borsària. El potencial d'aquesta tècnica fa que constantment surtin noves estructures i noves aplicacions. Dins aquestes novetats, les *generative adversarial neural networks* són un tipus de xarxa molt estudiada últimament per la seva capacitat de generar dades, inicialment imatges però també altres formes com poden ser seqüències temporals.

A més a més, en l'àmbit financer, també fa poc temps que s'intenten implementar mètodes matemàtics per predir el comportament dels mercats. Les sèries temporals complexes, com les dels mercats financers, són especialment difícils de modelitzar pel seu comportament estocàstic. La dificultat creix en incrementar la freqüència de mostreig, com és el cas de les dades en alta freqüència. Ara bé, l'interès també creix perquè treballar amb major freqüència permet realitzar més operacions i, conseqüentment, obtenir més beneficis.

És per tant la combinació d'aquests dos elements el motor que motiva aquest projecte que pretén observar l'eficiència de les xarxes neuronals generatives antagòniques en mercats d'alta freqüència.

1.2 Requeriments previs

En aspectes teòrics, per treballar amb xarxes neuronals és necessari tenir una base de càlcul, àlgebra i estadística, ja que és una tècnica que engloba aquestes tres disciplines. A efecte de poder implementar els algorismes, són necessaris coneixements de programació, sobretot de Python, ja que és el llenguatge amb major nombre de llibreries destinades a xarxes neuronals.

En aspectes més pràctics, és important comptar amb un ordinador potent que tingui força capacitat de còmput. Cal dir que en el procés d'entrenament d'una xarxa neuronal es realitzen moltes operacions i cal tenir un sistema que ho pugui suportar i que, alhora, sigui capaç de realitzar-lo el més ràpid possible. El cert és que es realitzen moltes iteracions i això fa que també sigui un procés costós en el temps. És recomanable tenir una bona memòria RAM i disposar d'una GPU per accelerar els processos. Ara bé, treballar en una GPU amb Python no és senzill. De fet, no es treballa directament en Python, sinó en Cython, una extensió del llenguatge a C++ per poder donar ordres a la targeta gràfica. Les comandes d'aquesta extensió estan molt ben implementades sobre targetes Nvidia, ara bé, presenten problemes en altres dispositius.

Per iniciar el projecte no es disposava dels requisits tecnològics necessaris. És per això

que la xarxa s'ha implementat amb l'ajuda de Google Colab, una plataforma al núvol que de forma gratuïta et permet executar codi en Python sobre una GPU.

1.3 Metodologia

Pel que fa a la metodologia seguida en la realització del projecte, s'ha seguit un procés que concorda amb l'ordre en què es van exposant els temes en el treball. En primer lloc, s'ha realitzat una recerca sobre les xarxes neuronals en general focalitzant-se en les estructures utilitzades en el projecte. Així doncs, en els primers capítols s'exposen els continguts i la informació adquirida en aquest procés.

Seguidament, s'ha realitzat un estudi del problema en el qual se centra el projecte, la predicció del comportament dels mercats en alta freqüència. En aquesta part del treball, es parla tant de la problemàtica i la situació actual, com de l'article sobre el qual parteix el projecte.

Prèviament a la implementació directa de la xarxa, s'ha realitzat una cerca de dades en les condicions desitjades. Així com una anàlisi d'aquestes per tal d'observar les diferències en el comportament en funció del temps de mostreig.

Per últim, s'ha procedit a la implementació de la GAN. El procés d'implementació ha seguit un sistema prova-error, és a dir, s'han anat realitzant canvis en la xarxa i analitzant els resultats per tal d'obtenir el millor funcionament possible.

1.4 Objectius

L'objectiu principal d'aquest projecte és realitzar un estudi sobre les *Generative Adversarial Neural Networks* i analitzar el seu funcionament sobre seqüències temporals. Concretament, es treballarà amb dades borsàries en alta freqüència mostrejades en intervals de nanosegons.

L'anàlisi borsària no és pas un terreny senzill i, de fet, les xarxes neuronals encara no han demostrat gaire èxit en ell. A l'inici de la recerca, com cal fer en tot projecte, l'objectiu és assolir un sistema funcional i capaç de predir el comportament del valor borsari escollit. Des de l'article sobre què s'inicia la recerca [36], s'assegura haver assolit un mètode millor que les tècniques de referència actual. Ara bé, també es mostren reservats amb la capacitat del model en altres mercats i espais temporals. És per això que en el projecte es pretén sobretot fer una anàlisi del comportament de les GAN en aquest tipus de problemàtica. És a dir, i atenent a les limitacions que comporta un estudi com aquest a nivell de grau, no es busca obtenir un nou mètode, sinó comprovar el funcionament de les GAN en situacions versemblants.

1.5 Abast

En el projecte es pretén analitzar la capacitat de les GAN per a predir el comportament del preu de les accions d'Apple en alta freqüència en el rang horari entre les 9:30:00 i les 10:30:00 del matí del 21 de juny de 2012.

No és objectiu de l'abast d'aquest projecte pretendre obtenir un mètode millor que els actuals, ni observar el comportament en altres mercats o en altres espais horaris.

Capítol 2

Conceptes bàsics

En aquest capítol, abans de començar amb l'exposició del treball, es realitzarà una breu explicació sobre el funcionament general i l'origen de les xarxes neuronals de forma introductòria. Així doncs, a continuació s'exposaran alguns conceptes bàsics i importants per poder seguir el desenvolupament del projecte. Aquest capítol és d'interès, sobretot, per aquells lectors que no tinguin una idea clara sobre que són les xarxes neuronals. Aquells qui ja tinguin coneixements de *Deep Learning* poden avançar al següent capítol.

2.1 Definició

Una xarxa neuronal, *Neural Network* en anglès, és un model matemàtic dissenyat per processar relacions no lineals entre una informació d'entrada (*inputs*) i una de sortida (*outputs*).

$$outputs = f(inputs) \tag{2.1}$$

Aquest model es basa en la combinació lineal de variables ponderades per un conjunt de pesos w_i. Posteriorment, a la suma dels valors ponderats se li aplica una funció *f(·)*, anomenada funció d'activació o *activation function*, per donar no linealitat a la xarxa. L'exemple més bàsic d'una xarxa neuronal és l'anomenat perceptró, ideat per Frank Rosenblatt l'any 1957 (veure Figura 2.1).

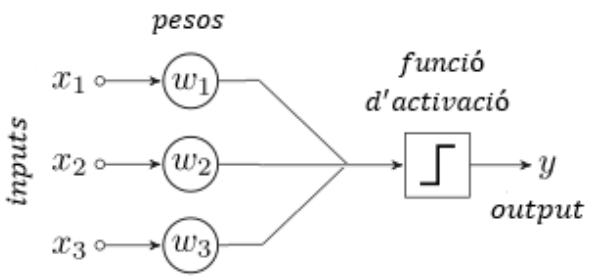


Figura 2.1: Diagrama d'un perceptró.

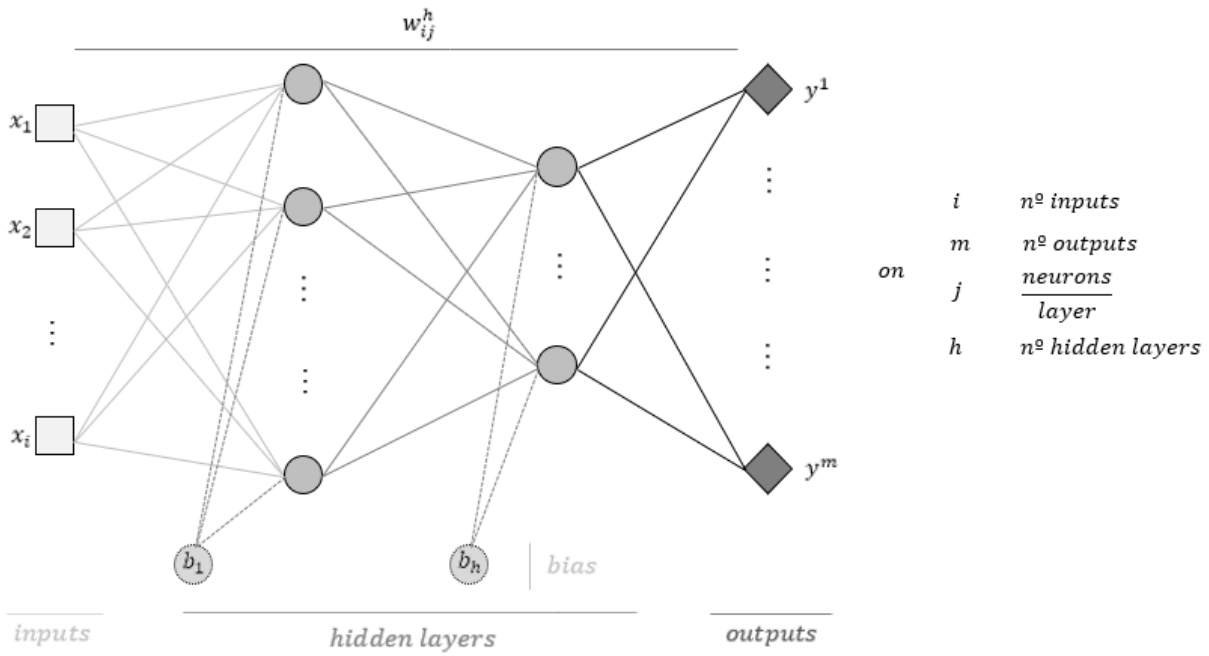


Figura 2.2: Diagrama d'una *feed-forward neural network*.

Es pot observar com la xarxa exposada en la Figura 2.1 representa la funció:

$$y = f(x_1 w_1 + x_2 w_2 + x_3 w_3) \tag{2.2}$$

En la representació gràfica de les xarxes neuronals, els punts on es combinen un conjunt de variables són coneguts com a neurones. Sovint també se suma una constant de biaix. En el perceptró només hi ha una neurona, però en general les xarxes són estructures més complexes que agrupen un nombre n de neurones. Les neurones, alhora, s'agrupen en capes, *layers*, i així es crea una estructura complexa on s'interconnecten diverses neurones. La presència de diverses capes és el que fa que les xarxes neuronals siguin una tècnica d'aprenentatge profund o *Deep Learning*. En la Figura 2.2 es pot observar una representació generalitzada d'una xarxa neuronal on, sigui $\hat{F}$ la funció d'activació, x_i el vector d'*inputs* i ϕ una matriu que agrupa tots els pesos w_{ij}^h, els *outputs* de la xarxa serien:

$$y^m = \hat{F}(x_i; \phi) \tag{2.3}$$

Concretament, la xarxa representada en la Figura 2.2 és el que es coneix com xarxa neuronal directa, en anglès *Feed-forward Neural Network*. S'anomena així perquè totes les variables que surten d'una mateixa neurona en una capa h, es combinen per arribar a una capa $h+1$. Aquest tipus d'estructures són l'exemple per excel·lència del funcionament de les xarxes neuronals. De fet, són l'evolució del perceptró. Actualment existeixen xarxes on les variables es mouen a capes anteriors, són les anomenades xarxes recurrents o *Recurrent Neural Networks (RNN)*, les quals es veuran en el Capítol 3. A nivell teòric i com a

introducció, les *feed-forward neural networks* són un exemple adequat per comprendre el funcionament d'una xarxa neuronal.

2.2 Aprenentatge

Si les xarxes neuronals són considerades una tècnica de *Machine Learning* és per la seva capacitat d'aprendre la funció que representen. El concepte d'aprendre fa referència a la capacitat de calcular el valor òptim dels pesos de cada neurona a partir d'un algoritme d'entrenament o *training*.

El primer pas per entrenar una xarxa és recollir exemples dels quals la xarxa pugui aprendre. Per això cal obtenir un conjunt de parelles de variables d'*inputs* i d'*outputs* corresponents. És a dir, en dissenyar una xarxa, s'ha de saber quins resultats y^m (*targets*) s'han d'obtenir donats uns determinats valors d'entrada x_i. Per exemple, si es vol una xarxa que donades 3 variables d'entrada retorni la suma de les dos primeres multiplicada per la tercera tal que:

$$y = (x_1 + x_2) \cdot x_3 \tag{2.4}$$

Llavors, un exemple de dades seria:

$$inputs \rightarrow \begin{bmatrix} 2 & 3 & 5 \\ -1 & 0.3 & 4 \\ 3 & 4 & 10 \\ -2 & -4 & -5.5 \end{bmatrix} \qquad outputs \rightarrow \begin{bmatrix} 25 \\ -1.2 \\ 70 \\ 33 \end{bmatrix}$$

Aquest tipus d'aprenentatge s'anomena *aprenentatge supervisat*, ja que per cada cas se sap la sortida que li correspon. Sol ser el tipus d'aprenentatge més comú. Tot i això, existeixen casos on no es coneix o és difícil recollir aquesta correspondència entre variables i s'opta per l'anomenat *aprenetatge no supervisat*. Un exemple en seria el que es coneix com *clustering*, on el sistema aprèn a partir de les distàcies en un cert espai entre les entrades.

El nombre de dades que es requereixen per entrenar una xarxa sol ser bastant elevat i augmenta en augmentar la complexitat de l'estructura. El més habitual és aconseguir un paquet de dades i fraccionar-lo en dues parts. Un primer paquet per entrenar la xarxa, *training set*, i un segon per comprovar el seu funcionament un cop entrenada, *test set*. La propoció de cada conjunt de dades sol ser 80 - 20 % del conjunt inicial.

Un cop definit el *training set* es pot començar el procés d'entrenament. El primer pas consisteix en calcular els resultats de la xarxa donades unes variables d'entrada, el que es coneix com propagació endavant. Seguidament cal calcular l'error dels resultats obtinguts respecte els resultats correctes. Per fer-ho, es defineix una funció d'error o *Loss Function*, també coneguda com funció de cost o d'aprenentatge. Amb l'error obtingut s'actualitzen

els pesos. Generalment, es solen utilitzar tècniques d'optimització basades en el descens pel gradient per tal de reduir l'error. En altres paraules, es calculen els gradients de l'error respecte cadascun dels paràmetres, pesos, i, amb aquests gradients, s'actualitzen els pesos per la següent iteració. El càlcul dels gradients es fa amb una tècnica coneguda com propagació enrere o *backpropagation*.

Per exemple, en el cas del perceptró de la Figura 2.1, sigui :

$$input = \begin{bmatrix} 2 \\ 4 \\ 5 \end{bmatrix} \qquad output = \begin{bmatrix} 25 \end{bmatrix} \qquad pesos = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.8 \end{bmatrix}$$

utilitzant una funció ReLU com a funció d'activació[1]:

$$f(x) = max(0, x) \tag{2.5}$$

la funció de mínim quadrat (MSE) com a *loss function*:

$$L(\hat{y}, y) = \frac{1}{2} \sum_{n=1}^{N} \|y_i - \hat{y}_i\|^2 \tag{2.6}$$

i un descens pel màxim pendent com a mètode d'optimització:

$$w_i^{'} = w_i - \eta \frac{\partial L}{\partial w_i} \qquad \text{on} \quad \eta = 0.02 \quad (learning\ rate) \tag{2.7}$$

aleshores, la iteració d'entrenament seria:

- Propagació endavant:

$$y = f(x_1 w_1 + x_2 w_2 + x_3 w_3) = f(2 \cdot 0.5 + 4 \cdot 0.3 + 5 \cdot 0.8) = 6.2 \tag{2.8}$$

- Càlcul de l'error:

$$L = \frac{1}{2} |25 - 6.2|^2 = 176.72 \tag{2.9}$$

- Càlcul dels gradients:

$$\frac{\partial L}{\partial \hat{y}} = -1 \cdot (y - \hat{y}) = -6.2 \tag{2.10}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial y}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot x_1 = -6.2 \cdot 0.5 = -3.1 \tag{2.11}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial y}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot x_2 = -6.2 \cdot 0.3 = -1.86 \tag{2.12}$$

[1] Veure Capítol 3.

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial y}{\partial w_3} = \frac{\partial L}{\partial \hat{y}} \cdot x_3 = -6.2 \cdot 0.8 = -4.96 \tag{2.13}$$

- Actualització dels pesos:

$$w_1^{'} = 0.5 - 0.02 \cdot (-3.1) = 0.562 \tag{2.14}$$

$$w_2^{'} = 0.3 - 0.02 \cdot (-1.86) = 0.3372 \tag{2.15}$$

$$w_3^{'} = 0.8 - 0.02 \cdot (-4.96) = 0.8992 \tag{2.16}$$

- Següent iteració.

En resum, cal entendre que, en una xarxa, per una banda es defineix l'estructura en què es relacionen les variables i les funcions d'activació i, per l'altra, es defineix com aprendrà la xarxa amb la funció de cost i el mètode d'optimització. Cal comentar que el procés d'entrenament sol ser un procés llarg i que requereix força capacitat de còmput, com es podrà veure al Capítol 3.

2.3 Origen i evolució

Un cop introduït el concepte de xarxes neuronals, toca fer un breu repàs per la seva història. Les xarxes neuronals són una branca del que es coneix com a *Deep Learning*, el qual és una branca del *Machine Learning* que, alhora, és un concepte que s'engloba en el marc de la Intel·ligència Artificial (AI). Actualment hi ha força confusió entre els tres conceptes. El cert és que des de fa aproximadament una dècada, les tècniques de *Deep Learning* han superat en eficiència qualsevol altra tècnica de *Machine Learning*. És per això que el gruix de la investigació científica es concentra en l'estudi de nous models de xarxes neuronals i el monopoli d'aquestes causa que algunes persones parlin de *Deep Learning* com a sinònim dels altres conceptes. L'eficiència de les xarxes neuronals no es deu només a l'obtenció de resultats més acurats, sinó també a la capacitat de modelar funcions complexes en diferents tipus de dades. Un dels avantatges que tenen és el fet de poder aprendre a extreure característiques o *features* de forma autònoma. Així doncs, a una xarxa neuronal se li pot subministrar directament *raw data*, mentre que en altres tècniques cal tractar les dades a l'inici.

Entrant en la història, tot i que les *Neural Networks* poden semblar un concepte molt modern, el cert és que el seu origen data de finals de la dècada dels 40 del segle passat. L'any 1943, els nordamericans McCulloch i Pitts van presentar el primer model de neurona [16] en un disseny molt similar al perceptró, presentada a la Figura 2.1. En aquella època també van destacar els estudis sobre aprenentatge del psicòleg Donald Hebb. No va ser però, fins al 1957, quan Rosenblatt va presentar el primer intent d'implementació d'una neurona de forma computacional amb el perceptró. Els anys 80 va ser una dècada prolífera

en la investigació en xarxes neuronals amb l'aparició de la tècnica de *backpropagation*, per exemple. Tot i que, la poca capacitat de còmput de la tecnologia de l'època va fer molt difícil assolir algoritmes eficients.

Va ser a partir dels anys 90 on la investigació en xarxes neuronals va tornar a repuntar. De fet, l'any 1997 es va produir una fita històrica en l'àmbit de la intel·ligència artificial quan l'algoritme d'IBM Deep Blue va aconseguir derrotar a Garry Kasparov en una partida d'escacs. El cert és que, en aquella època, la tecnologia estava evolucionant amb força i cada vegada eren més altes les capacitats de còmput. Això permetia l'augment de les dimensions de les xarxes en major nombre de capes, *hidden layers*, i, conseqüentment, major eficiència. A partir de l'any 2010, la recerca en xarxes neuronals augmenta de forma exponencial. De fet, diàriament es publiquen nous articles sobre *Deep Learning*. Les grans potències tecnològiques van començar a centrar esforços a millorar el rendiment dels seus sistemes de processament de dades i van crear àrees específiques centrades en el desenvolupament de xarxes neuronals. Algunes empreses també van desenvolupar les seves pròpies llibreries de *Neural Networks*. Els casos més coneguts serien el TensorFlow de Google o el PyTorch de Facebook. El creixement de les capacitats en l'estudi de les xarxes neuronals es fa visible en observar l'evolució dels *datasets*. Prop del 1950, va sorgir el famós Iris dataset amb unes dimensions aproximades de 10^2 dades. En contrast, els *datasets* originats a partir del 2000, com MNIST, CIFAR-10 o ImageNet, tenen dimensions superiors a 10^4 dades i, a més a més, les matrius de dades per cada patró d'entrenament són de grans dimensions, ja que són imatges. També ha augmentat el nombre de connexions per neurona. El 1960 no superaven les 10 connexions mentre que, actualment, per exemple, la xarxa GoogLeNet ja en té prop de 10^4, gairebé igual a un cervell d'un gat.

En l'actualitat les principals àrees de recerca en *Deep Learning* són en el camp de la visió per computador (*Computer Vision*), amb l'ús de xarxes convolucionals, i en el processament de text (*Natural Language Processing NLP*), amb l'ús de xarxes recurrents. Paral·lelament, ha crescut amb força l'estudi de xarxes generatives, les quals són el focus d'estudi d'aquest projecte. També són centre d'estudi les tècniques de *Reinforcement Learning* i, recentment, les *Graph Neural Networks*, tècniques que queden fora la visió d'aquest treball. Per més informació sobre *Deep Learning* i xarxes neuronals, es recomana llegir el llibre de Ian Goodfellow, Yoshua Bengio i Aaron Courville [8], tres dels principals investigadors en el tema juntament amb Geoffrey Hinton i Yann LeCun.

Capítol 3

Xarxes neuronals: estat de l'art

En aquest capítol s'exposarà l'estat de l'art dels diferents elements que componen una xarxa neuronal fins a la realització del treball. Tal com s'ha comentat, la investigació en *Deep Learning* ha crescut els últims anys i hi ha nous avanços gairebé de forma diària. Per això, les següents tècniques exposades se centraran en les tècniques utilitzades en el mateix projecte i no s'exposaran els conceptes que quedin fora aquesta visió.

Per millorar la comprensió, els conceptes s'ordenaran seguint el procés de disseny d'una xarxa. Així doncs, primer es parlarà de les estructures utilitzades i de les funcions d'activació. Seguidament s'explicaran les funcions de pèrdua i l'optimització dels paràmetres. I, finalment, es conclourà amb l'exposició d'altres conceptes rellevants pel bon funcionament de la xarxa.

3.1 Xarxes Convolucionals

Les xarxes convolucionals, també anomenades ConvNets o CNN, són un model molt utilitzat en visió per computador gràcies a la capacitat que tenen per tractar les imatges. El model s'inspira en el sistema de visió dels mamífers i en com el seu cervell analitza les imatges. Aquest concepte s'origina el 1980 amb el Neocognitron presentat per Kunihiko Fukushima [5] . No és fins al 1998 quan Yann LeCun [18] perfecciona la tècnica i, aquesta, comença a popularitzar-se fins a l'actualitat. El nom de la xarxa es deu a l'operació característica que utilitza per filtrar les dades, la convolució.

3.1.1 Definició de convolució

La convolució és una operació que mostra la quantitat de superposició que hi ha en una funció quan aquesta es desplaça sobre una altra. Com es pot observar en la Figura 3.1, una funció $f(t)$ es mou contra una funció $g(t)$. Només quan ambdues funcions es troben parcialment superposades la convolució és no nul·la.

En una ConvNet, la convolució és el primer pas de la xarxa i es realitza entre matrius.

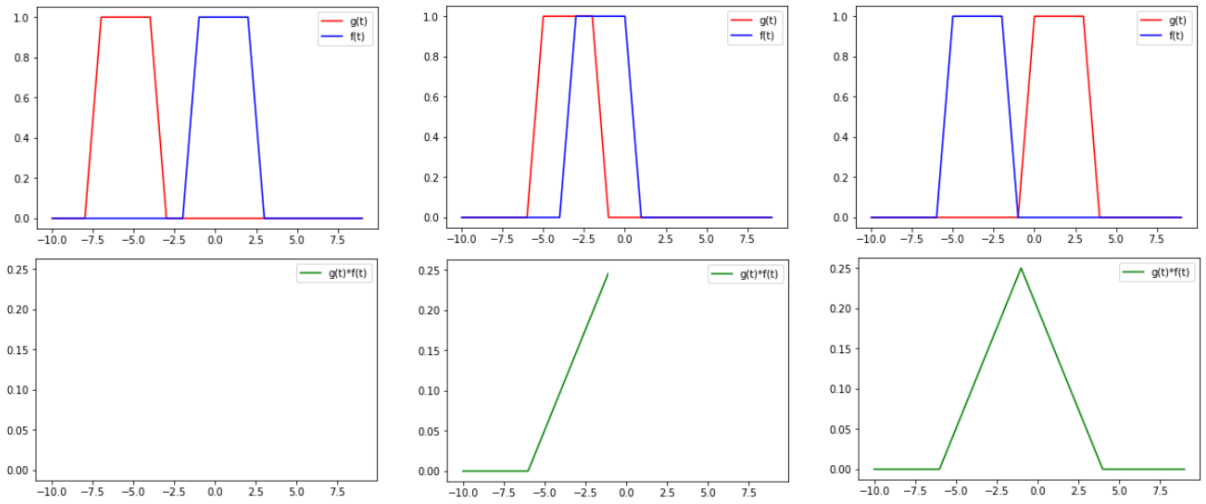


Figura 3.1: Representació de la convolució, en verd, entre $g(t)$, en vermell, i $f(t)$, en blau.

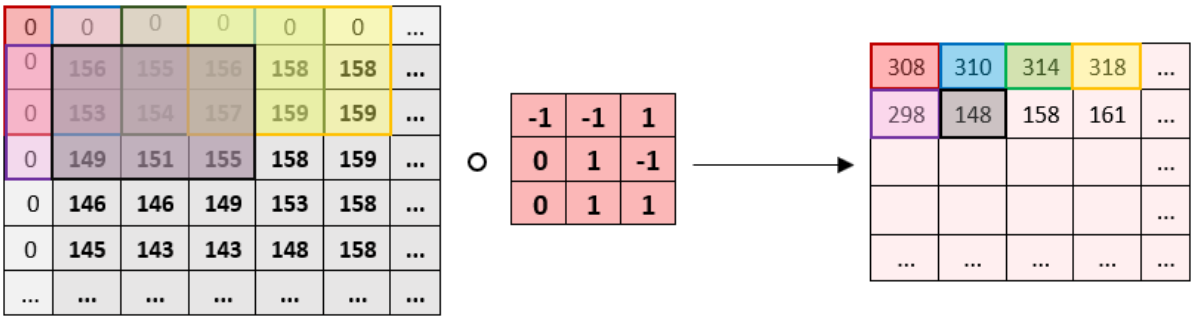


Figura 3.2: Representació de la convolució, entre una imatge $\mathbf{X}$ i un filtre $\mathbf{W}$.

Sigui $\mathbf{X}$ una matriu quadrada de dimensions $n \times n$ que representa els píxels d'una imatge, i $\mathbf{W}$ un matriu quadrada de dimensions $k \times k$ que representa un filtre, on $n > k$. Tal i com es pot veure en la Figura 3.2, el filtre recorre la imatge, generant una nova matriu $\mathbf{X}'$.

Matemàticament, la convolució és la suma dels valors resultants d'un producte d'Hadamard, terme a terme, entre dos matrius. Per exemple,

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 156 & 155 \\ 0 & 153 & 154 \end{bmatrix} \circ \begin{bmatrix} -1 & -1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{bmatrix} = 156 - 155 + 153 + 154 = 308 \tag{3.1}$$

El filtre o *kernel*, és la matriu que conté els pesos que la xarxa ha d'aprendre, per tant, és la base de l'estructura. Els valors solen iniciar-se de forma aleatòria o en base a filtres de xarxes amb una funció similar prèviament entrenades, el que es coneix com transferència de coneixement. Cal comentar també que, com s'observa en la Figura 3.2, la matriu $\mathbf{X}$ té un contorn de valors nuls, els quals s'han afegit premeditadament. Aquesta tècnica s'anomena *padding* i serveix per ajustar les dimensions dels *inputs*.

Un cop feta la convolució, se sol aplicar una funció d'activació a cada valor de la nova

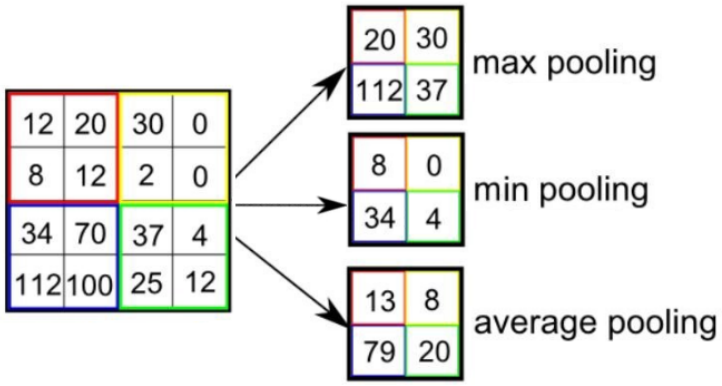


Figura 3.3: Representació de les diverses opcions més freqüents de *pooling*.

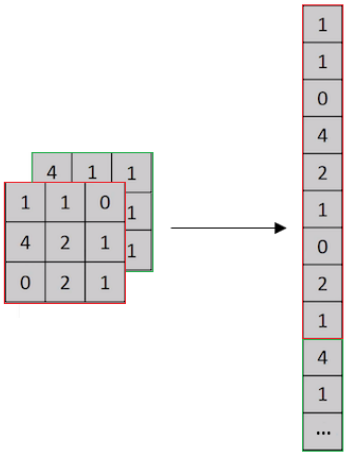


Figura 3.4: Representació de l'operació de *flattening*.

matriu $\mathbf{X}'$. Habitualment s'utilitza una ReLU, veure Secció 4 d'aquest Capítol. Com treballar amb imatges sol ser un procés molt costós pel que fa a capacitat de còmput, entre convolució i convolució, s'apliquen tècniques de reducció de dades, anomenades *pooling*. Aquestes consisteixen a agrupar els valors de la matriu $\mathbf{X}'$ en submatrius de mida $p \times p$. De cada submatriu s'escull només un valor i es crea una nova matriu $\mathbf{X}'_p$. Existeixen diversos criteris de selecció (veure Figura 3.3), d'entre els quals destaca el *max-pooling* que consisteix a escollir el valor més gran.

Aquest procés és la base de qualsevol xarxa convolucional. En una implementació real, el procés es realitza més d'una vegada i treballant amb més d'un filtre a la vegada en cada convolució. Com es pot veure a la Figura 3.5, un cop acabades les convolucions, s'obté un conjunt de matrius derivades dels filtres anteriors. Per tal d'obtenir informació d'elles, s'aplica el que es coneix com a *Flatten layer* (veure Figura 3.4), on els valors de les matrius s'uneixen en un sol vector de longitud l.

Amb les dades en forma de vector, es treballa com si es tractés d'una *feed-forward neural network* combinant els resultats en neurones. Aquesta última capa sol rebre el nom de *fully connected layer* o *dense layer*.

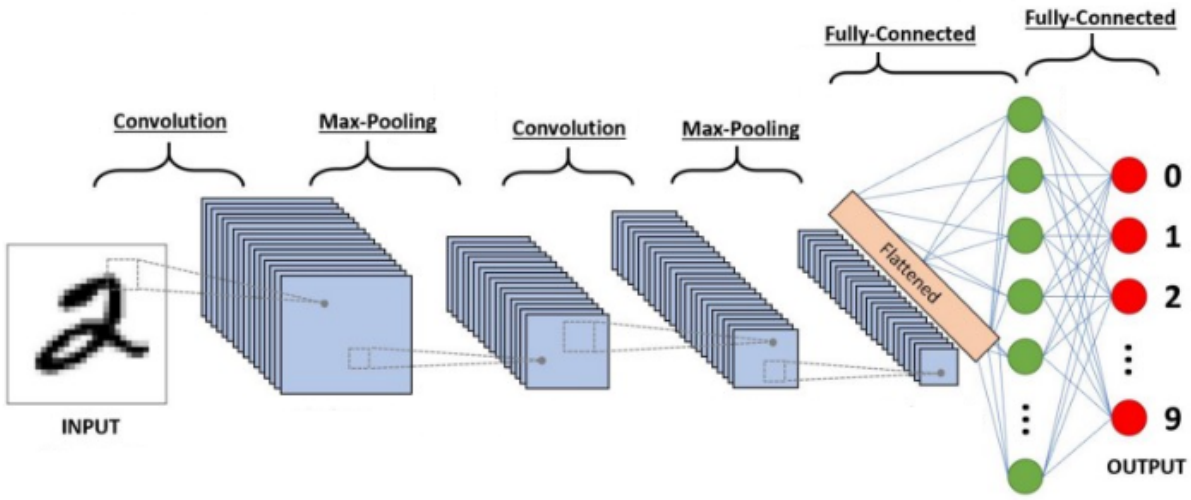


Figura 3.5: Representació d'una xarxa convolucional estàndard.

3.1.2 Discussió

Les xarxes convolucionals han suposat un gran avanç en les tècniques de reconeixement d'imatge. Dins el marc de les *Neural Networks*, la visió per computador ha anat augmentant el seu pes fins al punt que les bases de dades més conegudes i treballades són d'imatges, com MNIST o CIFAR-10. Les CNN han millorat qualsevol altre algoritme existent per tractar imatges i, tot i que el procés d'aprenentatge és llarg, l'execució de les xarxes convolucionals és relativament ràpid en comparació a les altres tècniques. Ara bé, ja s'han començat a observar els límits de les ConvNets. El principal detractor de l'actual model de xarxes convolucionals és el professor de la Universitat de Toronto Geoffrey Hinton. Des de fa ja un temps, el reconegut investigador anglès ha criticat les mancances d'aquest tipus d'estructures [11]. Fins i tot ha arribat a dir que el més sorprenent de les CNN no és el que poden arribar a fer, sinó el fet que funcionin tot hi tenir certs errors en el seu plantejament.

L'exemple més clar d'aquestes mancances es troba en la forma d'extreure *features* del model. El fet és que les xarxes convolucionals aprenen a detectar característiques, però no la relació entre elles. Per tant, són molt sensibles als canvis en l'orientació de les imatges. Aquest és un dels motius pel qual cal entrenar aquest tipus de xarxes amb un gran volum d'imatges perquè sigui capaç de detectar característiques en diferents perspectives. Per exemple, si s'observa la Figura 3.6, una xarxa convolucional detectaria en ambdós casos una cara humana, ja que la xarxa s'entrena per detectar dos ulls, un nas i una boca, i no pas la relació que hi ha entre els elements.

3.1.3 Actualitat

En l'actualitat, la recerca en xarxes convolucionals es focalitza en dos àmbits principals. Per una banda, se segueix treballant per millorar les tècniques de *Computer Vision* i

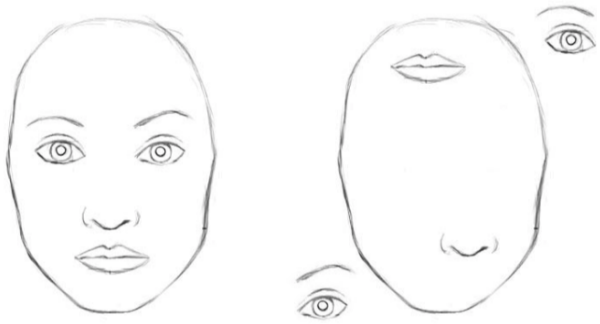


Figura 3.6: Representació d'una cara humana (esquerra) i d'una cara desordenada (dreta). Font [24].

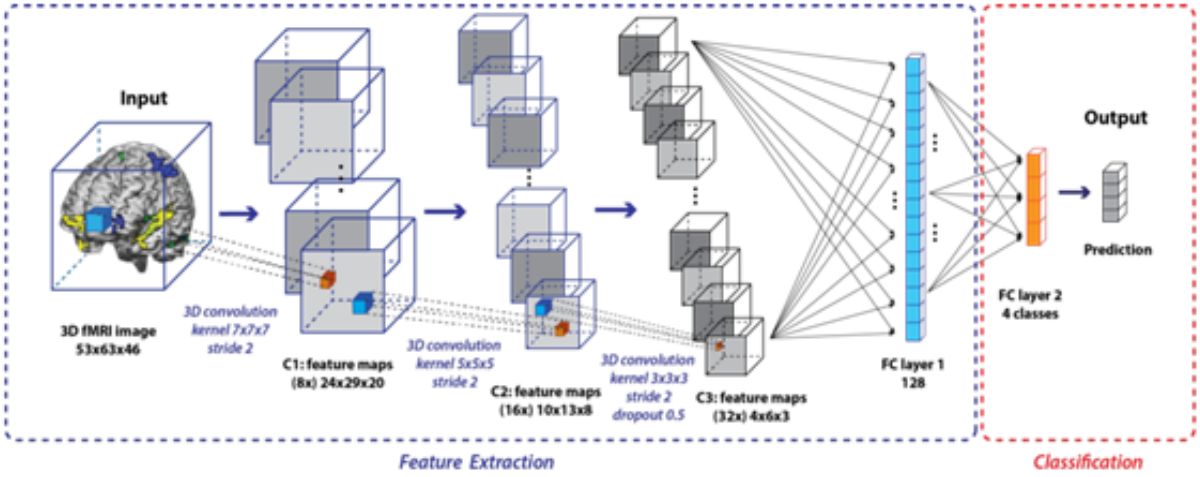


Figura 3.7: Exemple d'ús d'una xarxa convolucional 3D per l'estudi d'un encefalograma. Font [33]

reconeixement d'imatges. Per l'altra, les CNN han començat a introduir-se en el camp del processament de text, concretament, en *Natural Language Processing* (NLP).

3D Convolutional Neural Networks

L'origen de les xarxes convolucionals va ser treballant amb imatges bidimensionals. El 2012 aproximadament, van aparèixer els primers articles que utilitzaven convolucions tridimensionals per treballar amb seqüències d'imatges amb la intenció de classificar accions humanes, *human action recognition*. Amb el temps, les eines de CAD i modelatge 3D també han evolucionat, podent arribar a dissenyar models realístics de diversos elements, des d'objectes a òrgans. En aquest àmbit ha crescut la implementació de 3D ConvNets.

Capsule Networks

Com a resposta als problemes en les xarxes convolucionals, Geoffrey Hinton va presentar el 2017 les anomenades Capsule Netowrks [12]. Les CapsNets són unes xarxes convolucionals capaces d'aprendre la relació entre característiques d'una imatge. D'aquesta forma,

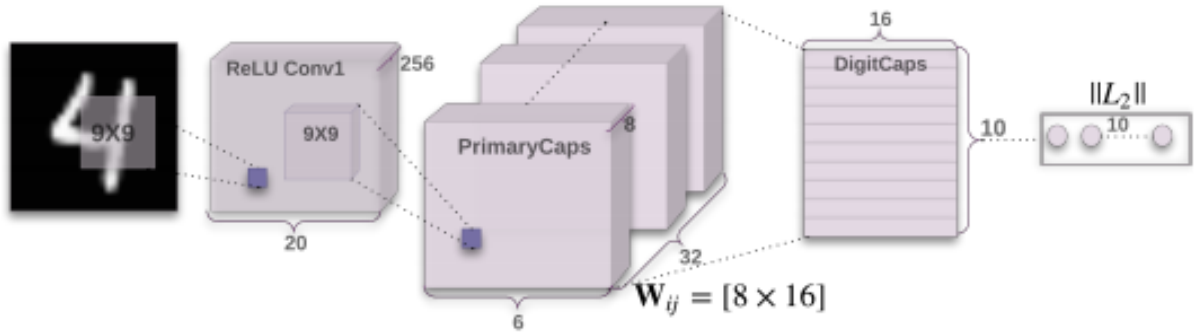


Figura 3.8: Plantejament inicial de l'estructura d'una Capsule Network. Font [12]

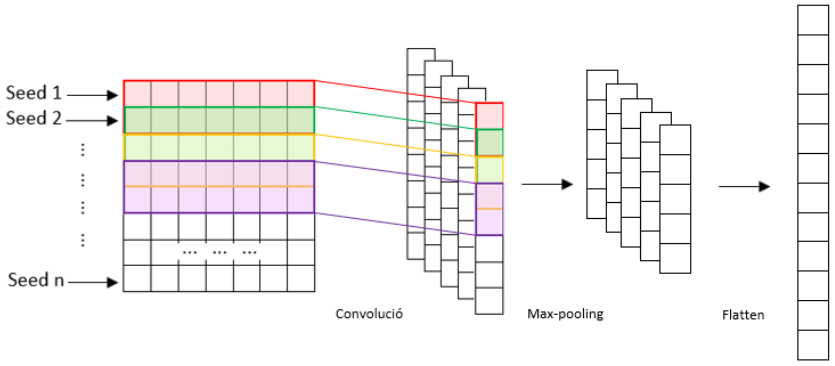


Figura 3.9: Convolució i *pooling* en una xarxa unidimensional.

la xarxa és robusta i capaç de detectar un objecte encara que aquest es trobi en perspectives diferents. Hinton aconsegueix aquesta capacitat eliminant els processos de *pooling*. En una Capsule Network, després de l'etapa de convolució, les matrius generades són empaquetades en forma de vectors en càpsules, d'aquí el seu nom. Aquest tipus de xarxes són molt interessants. De fet, ja s'ha demostrat que són més eficients que els ConvNets. Tot i això, el seu desenvolupament queda fora la visió del treball.

1D Convolutional Neural Networks

Així com existeixen les xarxes convolucionals 2D i 3D, si la convolució es realitza en un espai unidimensional, parlem de xarxes 1D convolucionals. Aquest tipus de xarxes han estat molt utilitzades els darrers anys per treballar amb *Natural Language Proccessing* i amb sèries temporals. De fet, en aquest projecte s'utilitzarà una xarxa 1D convolucional per treballar amb dades borsàries, les quals són precisament una sèrie temporal. Fora de les dimensions de la xarxa, els processos que implementa una 1D CNN no difereixen dels explicats anteriorment en el model 2D.

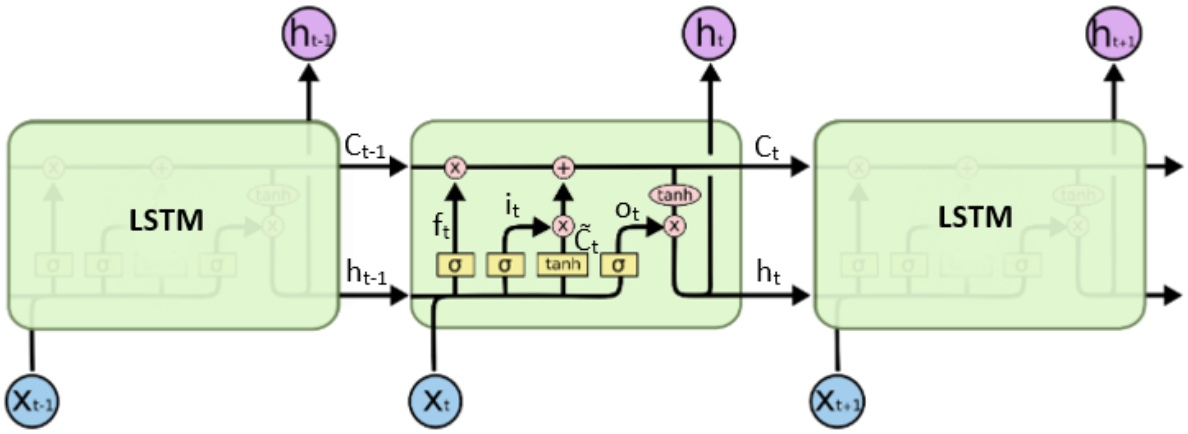


Figura 3.10: Diagrama de l'estructura d'una cel·la d'una Long Short-Term Memory.

3.2 Long Short-Term Memory Recurrent Networks

Les Long Short-Term Memory, o LSTM, són un tipus de xarxes recurrents. A diferència de les *feed-forward neural networks* vistes en el Capítol 2, les *recurrent neural networks* són un tipus de xarxes on cada neurona pot enviar informació a altres neurones del mateix estat o d'estats anteriors, és a dir, la propagació no és únicament directa. Les LSTM van ser introduïdes per Sepp Hochreiter i Jürgen Schmidhuber l'any 1997 [13] i, des d'aleshores, s'ha popularitzat el seu ús en l'estudi de seqüències, temporals i de text.

3.2.1 Definició

Les LSTM es defineixen en cel·les. Dins de cada cel·la es realitzen un conjunt d'operacions les quals es poden agrupar en quatre neurones. En la Figura 3.10 es pot observar l'estructura d'una cel·la [23]. Les cel·les s'agrupen en capes i, dins de cada capa, cada una de les cel·les transfereix informació a la cel·la següent, d'aquesta forma, si ho pensem de forma temporal, cada cel·la representa un estat en una seqüència.

En la Figura 3.10, X_t és un vector que actua com a *input* en la cel·la de l'estat t i h_t actua com a *output* de la cel·la. Alhora, es poden observar dos *inputs* més. Per una banda, h_{t-1}, conegut com a *hidden state*, és el resultat obtingut en l'estat anterior. És el que es coneix com a *Short-Term memory*, ja que dóna informació de l'últim estat de la seqüència. Per altra banda, C_{t-1} és el que es coneix com a *Long memory*. C_t, coneguda com a variable de context o *Cell state*, és una variable que recorre totes les cel·les d'una mateixa capa actualitzant-se en cada una d'elles. Aquesta variable intenta transmetre informació des de la primera a l'última cel·la, per això es diu que transporta informació de llarga durada.

Dins de la cel·la es poden identificar quatre variables. Aquestes es coneixen com a portes o *gates*. D'esquerra a dreta, la primera d'elles és la *forget gate* (f_t). Aquesta funció calcula un valor que multiplicarà la variable de context, per tant, el resultat de f_t indica

en quina mesura es conservarà la variable C_{t-1}, és a dir, si cal 'oblidar' o no el context. La funció de *forget gate* s'implementa tal que,

$$f_t = \sigma(W_{if}x_t + W_{hf}h_{t-1} + b_f) \tag{3.2}$$

Seguidament, la següent funció és l'anomenada *input gate* (i_t). Aquesta funció treballa directament amb els *inputs* de la cel·la, per tant, dóna informació sobre en quina quantitat cal considerar la informació que entra en l'estat t respecte la *long memory*. La funció i_t s'implementa tal que,

$$i_t = \sigma(W_{ii}x_t + W_{hi}h_{t-1} + b_i) \tag{3.3}$$

El resultat de la *input gate* actua com a factor ponderant de la variable $\widetilde{C}$. Aquesta variable computa la informació de l'estat t que cal afegir a la variable de context. Es calcula com,

$$\widetilde{C} = tanh(W_{ic}x_t + W_{hc}h_{t-1} + b_c) \tag{3.4}$$

Es pot notar que aquesta funció utilitza una funció diferent a les anteriors. El comportament d'aquesta es veurà en la Secció 4 d'aquest Capítol. Tot i això, es interessant considerar que la funció tanh(), a diferència de la sigmoide, pot oferir valors negatius. Per tant, la informació $\widetilde{C}$ s'afegirà o, contràriament, es traurà de la variable de context. Així doncs,

$$C_t = f_t \cdot C_{t_1} + i_t \cdot \widetilde{C}_t \tag{3.5}$$

Finalment, queda la funció *output gate* (o_t). Aquesta, indica quina informació és important considerar per obtenir un resultat i es calcula tal que,

$$o_t = \sigma(W_{io}x_t + W_{ho}h_{t-1} + b_o) \tag{3.6}$$

En aquest punt, el valor de C_t flueix al següent estat. Per obtenir el resultat de la cel·la es combina la variable de context amb la sortida de l'*output gate*.

$$\hat{y} = o_t \cdot tanh(C_t) \tag{3.7}$$

En funció de la seqüència amb la que es treballa es realitza una combinació específica de cel·les LSTM com l'exposada anteriorment. Existeixen tot tipus de combinacions que es poden classificar en funció del nombre de sortides i entrades que té la xarxa. En la Figura 3.11 es poden observar les estructures més freqüents.

3.2.2 Discussió

Com s'ha comentat en el Capítol 2, les xarxes neuronals actualitzen els seus pesos a partir del càlcul dels gradients. Com es veurà a la Secció 5 d'aquest mateix Capítol, calcular

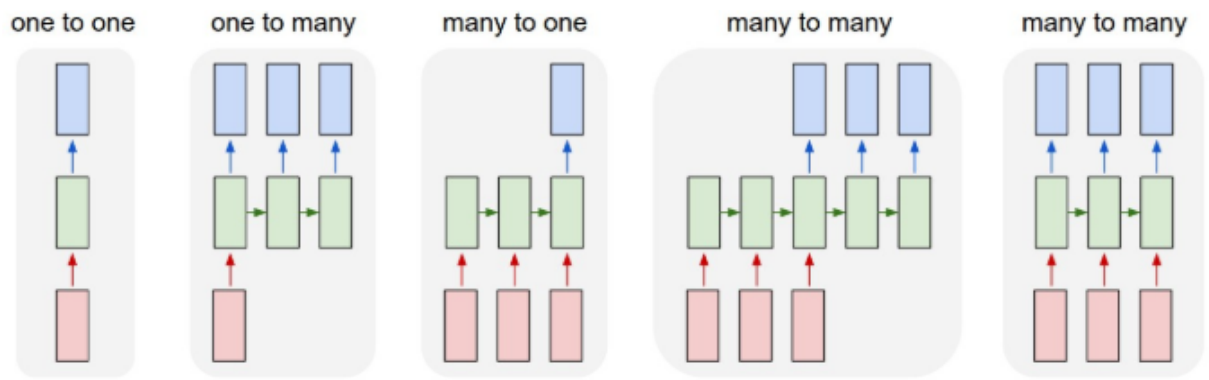


Figura 3.11: Representació de diferents estructures de xarxes LSTM. Font [28]

els gradients dels diferents paràmetres d'una xarxa no és pas una feina senzilla i implica un conjunt de multiplicacions de diferents termes. Aquest fet és el que causa un dels principals problemes dels models actuals. Per una banda, si els valors són molt elevats, el fet de multiplicar-los entre si fa que els gradients surtin molt elevats. Aquest fenomen es coneix com a *exploding gradients*. Per l'altra, si els valors són inferiors a 1, els gradients poden tendir a decréixer fins a anul·lar-se. El que es coneix com a *vanishing gradients*.

Abans de l'aparició de les LSTM, la xarxa recurrent per excel·lència era la Vanilla RNN. Una xarxa també composta per cel·les, però en cada cel·la només es realitzava una combinació lineal entre l'*input* de l'estat t i l'*output* de l'estat $t-1$. Aquesta cel·la tenia certs problemes d'eficiència i, entre aquests, en destacava precisament el de *vanishing gradients*. Les cel·les LSTM intenten solucionar aquest problema implementat l'esmentada variable de context o *cell state*. D'aquí en ve la seva popularitat. De fet, des del seu sorgiment, han estat el tipus de xarxa recurrent més utilitzada i estudiada.

Tot i això, les LSTM tenen certes mancances. En primer lloc, tot i millorar l'actuació de les Vanilla RNN, tampoc acaben de solucionar el problema dels *vanishing gradients*. De fet, aquest tipus de xarxes presenten problemes a l'hora d'intentar extreure característiques de llargues seqüències. Entre les comunitats de programadors s'estima que el màxim nombre de cel·les per capa per obtenir un bon rendiment es troba al voltant de 30. Per altra banda, les xarxes LSTM tenen un important problema de rendiment en el procés d'entrenament. El fet de tenir diverses operacions a cada cel·la i el fet de propagar-se la informació de cel·la en cel·la fa que aquest sigui molt lent en comparació a altres xarxes.

3.2.3 Actualitat

En l'actualitat, la recerca en Long Short-Term Memory Networks segueix vigent. De fet, segueixen sent el model més usat per tractar dades unidimensionals. Ara bé, han aparegut algunes variants focalitzades sobretot a solucionar el problema de la velocitat d'execució i la capacitat per aprendre de seqüències llargues.

Seqüències truncades

Una tendència freqüent és la de treballar amb un nombre determinat de cel·les LSTM t i fraccionar la seqüència d'entrada perquè s'adapti al nombre de cel·les. És a dir, s'entrenen les cel·les de forma iterativa amb diferents fragments de la sèrie d'*inputs* [2]. Aquí apareix el concepte d'*statful cells*. És a dir, el *hidden state* i el *cell state* de l'última capa es guarda i s'utilitza com a h_{t-1} i C_{t-1} de la primera cel·la a l'iniciar l'entrenament amb la següent part de la seqüència.

Estructura Encoder-Decoder

Un encoder és un tipus de model de *Deep Learning* que redueix la dimensió de les dades que li entren. Els *outputs* de l'etapa de codificació de les dades s'anomena espai latent o *latent-space representation*. I, complementàriament, un decoder és un model que agafa aquest estat reduït d'informació i el torna a la dimensió original [14]. La combinació d'aquestes dues estructures és el que es coneix com a *Autoencoder*, AE. Existeixen diferents formes de crear un autoencoder. La forma més senzilla seria mitjançant una combinació directa de neurones, com si fos una *feed-forward neural network*. Una altra variant seria utilitzar una convolució i una deconvolució.

Utilitzant aquest mètode, es pot reduir la seqüència de dades original a una seqüència que s'adapti a la dimensió de la xarxa LSTM. I, posteriorment, transformar les dades de sortida de la xarxa a la dimensió que es requereix.

Gated Recurrent Units

La forma en què s'interconnecten les operacions en una cel·la LSTM presenta variacions en la seva implementació. Hi ha variants que eliminen portes o d'altres on canvia la forma de modificar la variable de context. La variant més coneguda és l'anomenada *Gated Recurrent Unit* o GRU, presentada per Kyunghyun Cho l'any 2014 [3]. En les GRU es canvia l'estructura interna de les cel·les per millorar-ne la velocitat. S'ha comprovat com aquests tipus de cel·les augmenten de forma considerable la velocitat d'aprenentatge, ara bé, ho fan en detriment de la precisió, la qual disminueix lleugerament.

En la Figura 3.12, es pot observar com el *cell state* és el mateix que el *hidden state* i com el nombre de portes s'ha reduït. Les portes es poden calcular tal que,

$$z_t = \sigma(W_{iz}x_t + W_{hz}h_{t-1} + b_z) \tag{3.8}$$

$$r_t = \sigma(W_{ir}x_t + W_{hr}h_{t-1} + b_r) \tag{3.9}$$

$$\widetilde{h}_t = tanh(W_{ih}x_t + W_{hh}h_{t-1}r_t + b_r) \tag{3.10}$$

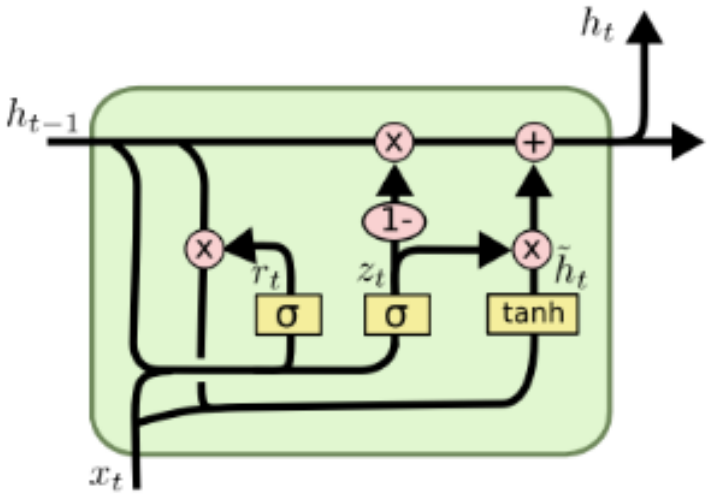


Figura 3.12: Representació d'una cel·la de GRU. Font [23]

Aleshores, la sortida de la cel·la és calcula com,

$$\hat{y} = (1 - z_t) \cdot \hat{y_{t-1}} + z_t \cdot \tilde{h}_t \tag{3.11}$$

1D Convolutional Neural Networks

En els últims anys les xarxes 1D convolucionals han anat robant terreny a les xarxes LSTM en l'estudi de dades seqüencials. S'ha demostrat la precisió de les 1D CNN en aquest tipus de problemes i, com a punt fort, les xarxes convolucionals són molt més ràpides d'entrenar. Avui dia, existeix una certa discrepància en la comunitat investigadora sobre si és millor un model o l'altre. Fins al moment, el més acceptat és que les xarxes LSTM obtenen resultats més exactes, ja que estan pensades per relacionar les variables d'una forma més coherent amb el problema en qüestió. Tot i això, alguns estudis apunten en direcció contrària. L'únic avantatge real de les LSTM sobre les 1D CNN, a part de l'opinió pública, és el fet que les xarxes LSTM són més flexibles a adaptar-se a rebre *inputs* de diferents dimensions.

3.3 Generative Adversarial Networks

Les xarxes generatives antagòniques, més conegudes com a *Generative Adversarial Networks* o GAN, són un tipus de xarxa introduïda per Ian Goodfellow l'any 2014 [9]. Aquest model, que és el focus d'estudi d'aquest treball, ha captat molta atenció els últims anys per la seva capacitat de generar dades d'una distribució determinada. És a dir, per exemple, si una GAN s'entrena amb imatges d'un gat, la xarxa serà capaç de generar imatges d'un gat a partir d'una llavor o *seed* de valors aleatoris. O, també, si una GAN s'entrena amb valors d'una seqüència, serà capaç de generar valors pertanyents a la mateixa

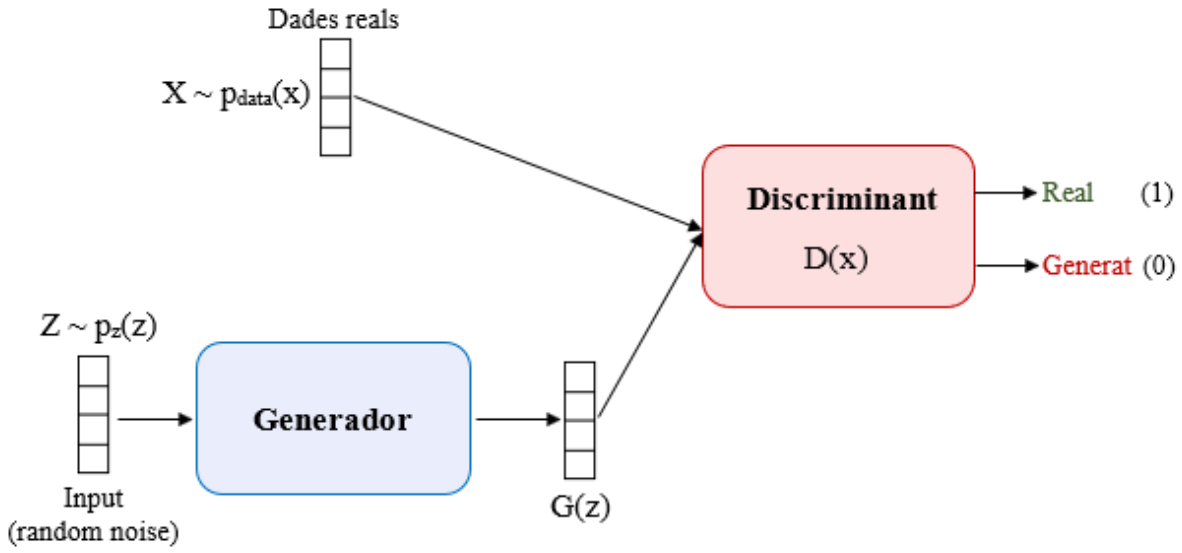


Figura 3.13: Representació d'una Generative Adversarial Network.

seqüència. La capacitat de generar valors implica intrínsecament una gran habilitat d'extreure característiques i aprendre relacions entre valors. Tot això, fa les xarxes generatives antagòniques un dels models amb més potencial actualment. De fet, l'investigador de Facebook AI, Yann LeCun, assegura que les GAN són la idea més interessant en l'última dècada en *Machine Learning*.

3.3.1 Definició

Les GAN són un tipus de model generatiu que es planteja com un joc entre dues xarxes neuronals. Una actua com a generador i té com a missió generar un conjunt de dades concretes. L'altra, actua com a discriminant i té la missió de distingir si un conjunt de dades són reals, és a dir, provenen del *training set*, o són falses, és a dir, són *outputs* del generador. En la Figura 3.13 es pot observar l'estructura comentada.

La interacció entre ambdues xarxes a l'hora d'entrenar-se és la clau del model. El mateix Goodfellow explica la relació entre Generador i Discriminant com si fossin un falsificador de bitllets (*counterfeiter*) i un policia. El Generador intenta generar cada vegada unes dades (bitllets) més fidels a la distribució desitjada (realitat), mentre que el Discriminant intenta cada vegada distingir millor la veracitat de les dades (bitllets). Per tant, com millor discrimini el Discriminant, millor haurà de generar el Generador, i viceversa. És per això, que a alguns investigadors no els hi agrada parlar de xarxes antagòniques, sinó de xarxes cooperatives, doncs una fa millor l'altra.

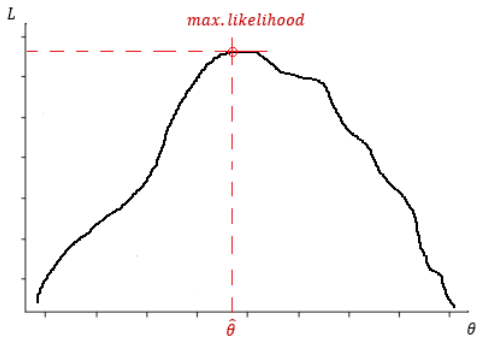


Figura 3.14: Representació del concepte de *maximum likelihood*.

Maximum likelihood

Com s'ha comentat, les GAN intenten generar dades semblants a una distribució real. Per tant, a diferència d'altres models on l'error es calcula amb funcions deterministes, aquí l'error es busca a partir de fórmules estocàstiques. L'elecció d'una correcta *loss function* és un tema de discussió que es tractarà més a fons en la Secció 5 d'aquest mateix capítol. Tot i això, de forma general, l'objectiu d'un model generatiu és obtenir el *maximum likelihood* d'un estimador (funció). És a dir, la màxima probabilitat que la distribució de les dades generades sigui la distribució de les dades reals. Cal dir que no tots els models generatius estan dissenyats per obtenir el *maximum likelihood*. De fet, les mateixes GAN no estan pas pensades per això, tot i que poden fer-ho. Aquesta generalització es fa amb intenció de poder discernir entre els diferents mètodes generatius. La idea és originària del mateix Goodfellow [7].

Per tant, partirem d'unes dades $x_1, x_2, ..., x_n$, de les quals en volem generar valors semblants. Aquestes pertanyen a una distribució $p_{data}(x|\theta)$ desconeguda. Aquesta distribució depèn d'uns paràmetres θ, que caracteritzen la distribució. Per exemple, una distribució normal es pot caracteritzar amb la mitjana i la variància. Aquests paràmetres són desconeguts, si els coneguéssim no caldria entrenar el model. És per això que els podem estimar com a $\hat{\theta}$. Paral·lelament, tenim un model que genera un conjunt de dades $\hat{x}_1, \hat{x}_2, ..., \hat{x}_n$ les quals volem saber si pertanyen o no a $p_{data}(x|\hat{\theta})$.

De la Figura 3.14, s'observa com en funció de les $\hat{x}_1, \hat{x}_2, ..., \hat{x}_n$, es pot calcular un valor de θ i, en funció del valor de θ, varia el valor de $L(\theta|\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$. La funció $L(\theta|\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$, seria la funció de cost elegida i indica la semblança entre la distribució real i la generada. El valor de θ que fa que $L(\theta|\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$ sigui el més semblant possible a la distribució $p_{data}(x|\theta)$, el *maximum likelihood*, és l'estimador més adequat pel nostre model, $\hat{\theta}$.

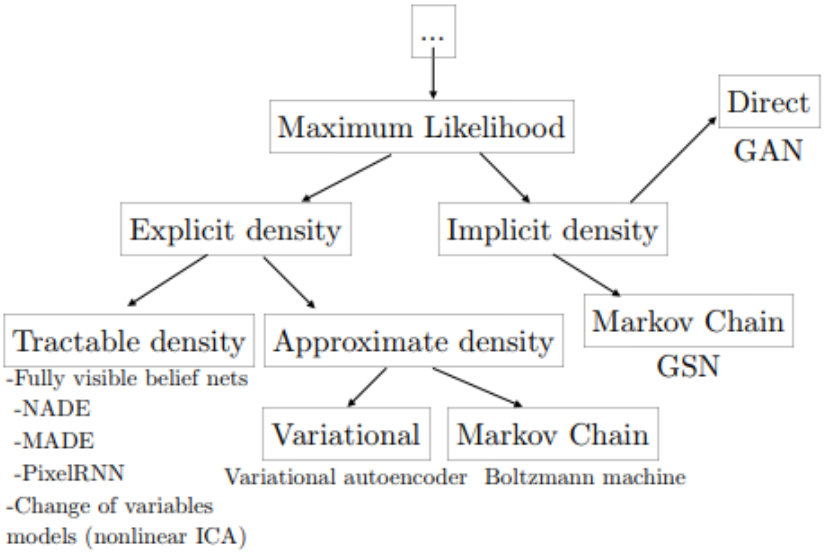


Figura 3.15: Classificació dels models generatius en funció del *maximum likelihood*. Font [7].

3.3.2 Discussió

Dins l'àmbit del *Machine Learning* existeixen diferents models generatius. Seguint la idea del *maximum likelihood* exposada anteriorment, una forma de classificar els models és en funció de com el calculen.

Explicit density models

De la Figura 3.15 se n'observen dues branques principals. En primer lloc, els models explícits. Els models explícits es basen en la prèvia definició de la distribució a imitar. Per exemple, suposem un jugador de futbol que ha xutat 6 penals en una temporada i n'ha marcat 5. Es vol modelitzar el nombre d'encerts com x. Com estem en un model explícit, cal suposar la distribució a què pertanyerà la variable,

$$x \sim Bin(n, p) \tag{3.12}$$

En aquest cas, n és el nombre d'intents (6) i és conegut. Mentre que p seria el paràmetre a estimar θ. Llavors, sigui la funció d'error, $L(\theta|\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$, el logaritme funció de probabilitat de la distribució, pmf en continu o pdf si és discreta, tal que,

$$L(p|x) = log\binom{n}{x} + log(p) + (n-x)log(1-p) \tag{3.13}$$

Es treballa en logaritmes perquè no canvia la posició del màxim i fa més senzill el procés de derivació. Així doncs, l'estimador de p seria,

$$\frac{\partial L(p|x)}{\partial p} = \frac{x}{p} + \frac{n-x}{1-p}(-1) = 0 \longrightarrow \hat{p} = \frac{x}{n} = \frac{5}{6} \tag{3.14}$$

Dins els models explícits en destaquen dos. En primer lloc, l'anomenat *Fully Visible Belief Nets* o FVBN. Una tècnica introduïda per Brendan Frey l'any 1998 i que és un dels models generatius més valorats. Les FVBN són la base de diversos prestigiosos algoritmes de DeepMind i es fonamenten en el fraccionament de probabilitats per intentar produir dades similars a la distribució referència. Les FVBN han obtingut bons resultats, tot i això, són un model lent.

L'altre gran model generatiu juntament amb les FVBN i les GAN és el *Variational Autoencoder* o VAE. Els VAE són un model amb estructura d'autoencoder, la qual s'ha exposat a la Secció 2, i són, probablement, el segon millor model generatiu darrere les GAN. Ara bé, són coneguts per la dificultat d'optimització que presenten. A més a més, en ser un model explícit, si no es defineix correctament la distribució que es vol aprendre, el model no funcionarà bé.

Implicit density models

Per altra banda, existeixen els models implícits on no s'estima prèviament la distribució a modelar. Aquest tipus d'algoritmes, com les màquines de Boltzmann per exemple, acostumen a treballar amb cadenes de Markov. Aquesta tècnica és encara molt utilitzada en diferents xarxes neuronals, però s'ha demostrat que és una tècnica que requereix certa capacitat de còmput i, a més a més, no treballa gaire bé en espais de moltes dimensions. És per aquest motiu que les GAN es van dissenyar per evitar l'ús de cadenes de Markov.

Respecte els models exposats anteriorment, les GAN han aconseguit evitar els desavantatges que presenten. Són capaces de generar dades en paral·lel (diversos *outputs* alhora) de forma que són més ràpides que les FVBN. No utilitzen cadenes de Markov a diferència dels altres models implícits. I, a priori, presenten una major capacitat d'aprenentatge que els VAE. A més a més, el motiu més important de la popularitat de les GAN és el fet que han sigut capaces d'obtenir major precisió en la generació de dades que qualsevol altre model. Cal explicar que no existeix un mètode objectiu per comparar els resultats d'un model generatiu, és per això, que l'afirmació de què les GAN generen millor que els altres models és fruit de la percepció i l'opinió de la comunitat investigadora.

Tot i això, les GAN també presenten els seus defectes. Concretament, el principal desavantatge d'aquest model es troba en el procés d'entrenament. Entrenar una GAN implica trobar un equilibri entre l'optimització dels paràmetres del Generador θ^G i l'optmització dels paràmetres del Discriminant θ^D. En teoria de jocs, es parlaria de trobar un equilibri de Nash, el qual és més complicat que optimitzar una funció independent.

De fet, les GAN, en ser un equilibri entre dues xarxes, solen tenir problemes per assolir una convergència. A més a més, els models generadors són complexes d'entrenar perquè poden arribar a col·lapsar, és a dir, aprendre a generar només una part de la distribució. És per aquests motius que gran part de la investigació actual se centra a buscar noves estructures per assolir major convergència.

3.3.3 Actualitat

En l'actualitat, el focus de la recerca en xarxes generatives antagòniques es troba en obtenir un algoritme d'entrenament robust que convergeixi en resultats correctes.

DCGAN

En l'apartat de definició del model, s'ha comentat que una GAN es basa en la interacció entre dues xarxes neuronals, però no s'ha especificat el tipus de xarxa. Això és perquè el model no es fonamenta en la xarxa que genera i la que discrimina, sinó en com s'entrenen una contra l'altra per obtenir millors resultats. Per tant, es pot utilitzar gairebé qualsevol classe de xarxa com a Generador i gairebé qualsevol com a Discriminant. Evidentment, en funció dels models que s'utilitzin, internament, la GAN tindrà una estructura o una altra. És per això que existeixen moltes varietats de GAN. Per exemple, una FF-GAN és una GAN que utilitza dues *Feed Forward neural networks*.

Dins aquest univers de varietats, l'estructura que ha causat més impacte fins al moment és la *Deep Convolutional GAN o DCGAN* [25]. Aquest model es va dissenyar per treballar amb imatges i presenta detalls en l'estructura que han servit d'inspiració per altres models. Per exemple, no utilitza capes de *pooling* després de la convolució, implementa capes de normalització entre capes per ajustar els valors interns de les xarxes o s'optimitza amb una tècnica anomenada ADAM. Més endavant es comentaran aquestes tècniques en major profunditat.

Addició de soroll

Alguns estudis apunten que el problema de convergència en les GAN pot venir del fet que el Generador no aprèn directament de les dades, sinó que aprèn a partir del Discriminant. Hi ha molta recerca en intentar variar la funció d'aprenentatge de les GAN, tal com s'ha comentat anteriorment. Una de les tècniques més comunes és implementar soroll, és a dir, dades distorsionades, en el procés d'entrenament. D'aquesta forma es busca evitar que la xarxa aprengui només a generar una distribució de dades concreta, el que es coneix com a col·lapse.

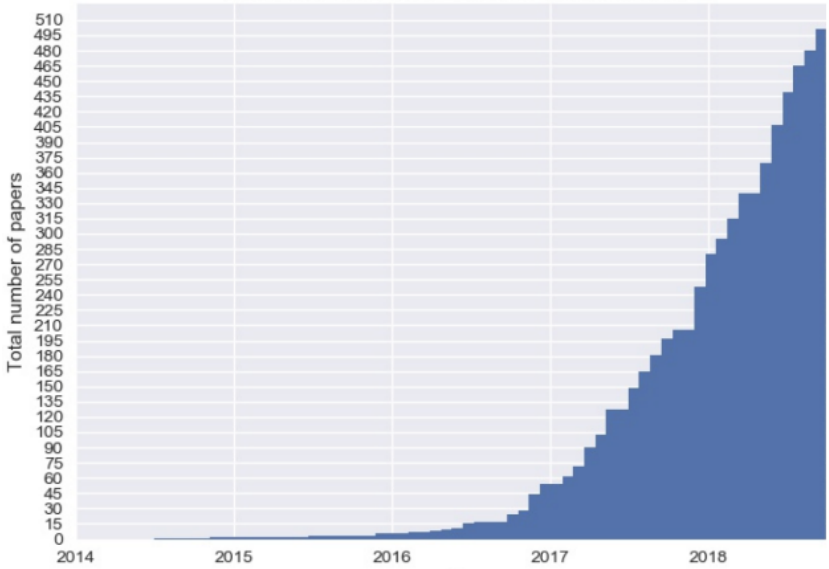


Figura 3.16: Evolució acumulada del nombre de GANs definides al llarg del temps. Font [30]

.

Wassertein GAN

Com s'ha comentat, l'estudi de noves mètriques i funcions de cost és el focus de la recerca actual. Existeixen una gran varietat d'alternatives [26], tant en funcions com en algoritmes. Ara bé, la variant més acceptada és l'anomenada *Wassetein GAN* o WGAN [1]. Les WGAN utilitzen la distància de Wassertein per mesurar l'error entre una distribució referència i una distribució generada. Estudis recents han demostrat que aquesta tècnica millora la convergència de la xarxa.

Reinforcement Learning

Existeixen altres variants que proposen utilitzar factors de penalització durant l'entrenament, com si fos *Reinforcement Learning*. Un exemple en seria una estructura combinada VAE-GAN proposada per investigadors de DeepMind [17].

En general, existeixen molts estudis sobre variables en la implementació de GANs (veure Figura 3.16) i, alhora, moltes discrepàncies. Alguns papers asseguren que cap model s'ha demostrat millor que l'original plantejat per Goodfellow, d'altres parlen de la importància de les dades per determinar quin algorisme és millor. El cert és que les GAN són bastant sensibles a modificacions. De fet, petits canvis en els *hyperparameters* poden ser molt problemàtics.

3.4 Funció d'activació

Com s'ha comentat anteriorment, les funcions d'activació serveixen per dotar de no linealitat a la xarxa i fer-la capaç d'entendre una major complexitat en les dades [32]. A més a més, també pot servir per adequar els *outputs* a la situació per la qual es requereixen. Per exemple, si es desitja fer una xarxa classificadora d'imatges en n classes diferents, aquesta ha de retornar la probabilitat que la imatge d'entrada pertanyi a cada una de les n classes. Com ha de retornar una probabilitat, el valor ha de pertànyer a l'interval [0,1], però la xarxa no ha de per què retornar un valor que hi pertanyi. En aquest cas, cal una funció d'activació que transformi els valors en el rang desitjat.

Una funció d'activació pot ser qualsevol funció. La seva elecció sempre dependrà del dissenyador, del tipus de xarxa i de l'objectiu que s'intenti assolir. A continuació s'exposen les funcions més comunes en el disseny de *neural networks*.

3.4.1 Sigmoide

La funció logística o sigmoide és una funció que retorna valors compresos entre 0 i 1, per tant, és útil per presentar probabilitats. Sol ser bastant utilitzada, sobretot, en problemes de classificació binària. Ara bé, la *sigmoid function* presenta dos principals inconvenients que cal conèixer. En primer lloc, la funció no està centrada al zero i això fa difícil el control dels gradients. En segon lloc, la funció no és pràctica per entrenar xarxes amb valors molt extrems doncs, si els valors són molt grans (o molt petits), la funció tendeix a retornar 1 (0). A l'hora de calcular el gradient, aquest fet implica que la derivada sigui avaluada en 1 (0) i, per tant, tingui un valor nul. Per tant, els gradients s'esvaeixen i la xarxa no s'actualitza. Aquest fenomen és un dels problemes més comuns en el disseny de xarxes neuronals, l'anomenat *Vanishing gradient problem* [34].

La funció logística és calcula tal que:

$$sigmoid(x) = \frac{1}{1 + e^{-\lambda x}} \tag{3.15}$$

Una variant de la funció sigmoide és l'anomenada *softmax*. Aquesta és una generalització de la funció logística molt utilitzada en multiclassificadors. La raó del seu ús és perquè, donades n classes, la funció *softmax* retorna n probabilitats tal que la seva suma és 1. Mentre que en el suposat d'usar una sigmoide, s'obtindrien n probabilitats cada una independent de la resta tal que el seu sumatori no resultaria en la unitat.

Per un vector z, la funció *softmax* es calcula tal que:

$$softmax(x) = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}} \qquad per \quad j = 1, ..., K \tag{3.16}$$

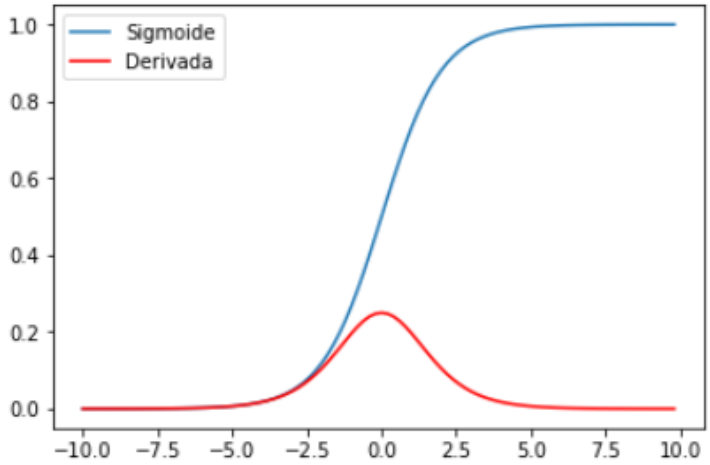


Figura 3.17: Representació d'una funció sigmoide.

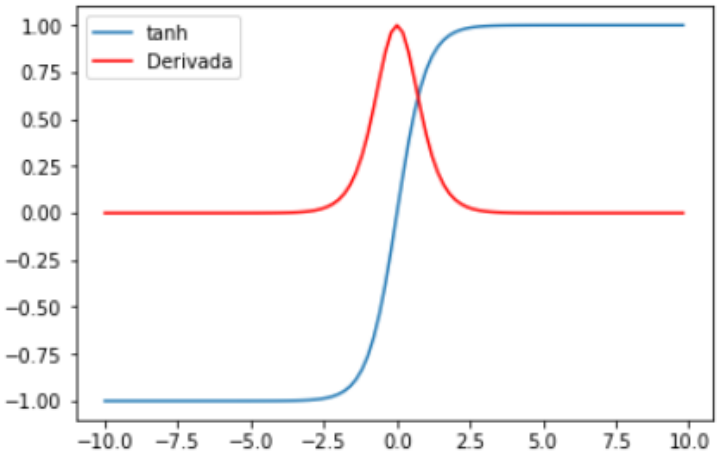


Figura 3.18: Representació d'una funció tangent hiperbòlica.

3.4.2 Tangent hiperbòlica

La funció tangent hiperbòlica o *tanh* és una actualització de la funció logística que retorna valors compresos entre -1 i 1. Per tant, la funció està centrada en el 0 a diferència de la funció sigmoide. Tot i això, la funció segueix patint problemes d'esvaïment, *vanishing*, per valors molt extrems. La capacitat de poder retornar tant valors positius com negatius és el gran valor d'aquesta funció, molt utilitzada en xarxes recurrents.

La funció *tanh* es calcula tal que:

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{3.17}$$

3.4.3 ReLU

La *Rectified Linear Unit* o ReLU és una funció que retorna 0 si els valors d'entrada són negatius i el valor d'entrada si és positiu. És una funció aparentment molt simple però és

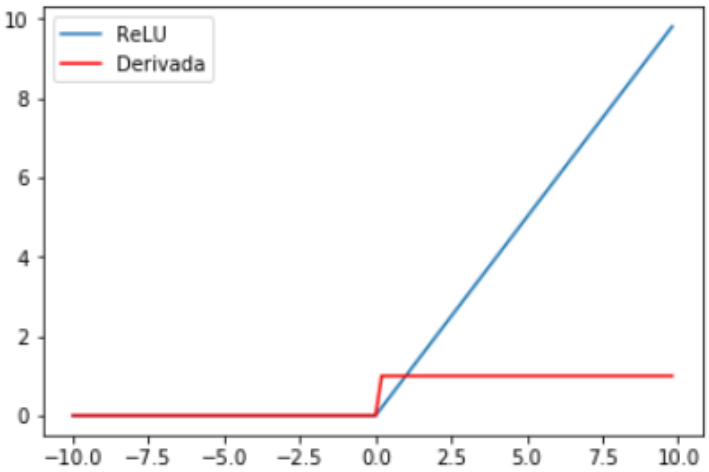


Figura 3.19: Representació d'una *Rectified Linear Unit*.

molt utilitzada i està adquirint força popularitat en les capes internes de les xarxes, més inclús que la pròpia sigmoide o tangent hiperbòlica. Els punts forts d'aquesta tècnica són que és molt ràpida a l'hora de processar-se i, segurament el més important, no presenta la possibilitat de saturar-se o esvair-se com si els hi passa a les dues funcions esmentades anteriorment.

La funció ReLU es calcula tal que:

$$relu(x) = max(0, x) \tag{3.18}$$

Un inconvenient que presenta la funció ReLU, és el fet d'anul·lar tots els valor negatius. Aquest tipus de funció d'activació va començar a usar-se en xarxes convolucionals per treballar amb imatges i, inicialment, en les capes internes no interessava treballar amb valors negatius. Actualment però, l'ús de les ReLU s'ha estès en altre tipus de xarxes on és interessant no oblidar els valors negatius. És per això que han sortit variants com, per exemple, la funció *Leaky* ReLU. En ella, els valors negatius no s'anul·len, sinó que descriuen una recta tal que:

$$leakyrelu(x) = \begin{cases} x & , si \quad x > 0 \\ \\ \alpha x & , altrament \end{cases} \tag{3.19}$$

Un altre inconvenient que s'ha observat últimament en la funció ReLU és la impossibilitat de derivar en el 0, ja que la funció canvia bruscament. És per això que s'ha dissenyat l'anomenada *SoftPlus* o SmoothReLU. Una funció pràcticament igual a la ReLU però amb un pas suau pel 0 tal que

$$SoftPlus(x) = log(1 + e^x) \tag{3.20}$$

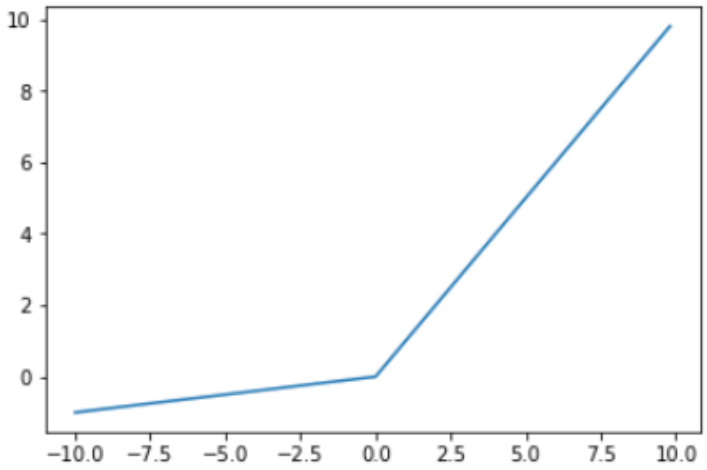


Figura 3.20: Representació d'una *Leaky* ReLU.

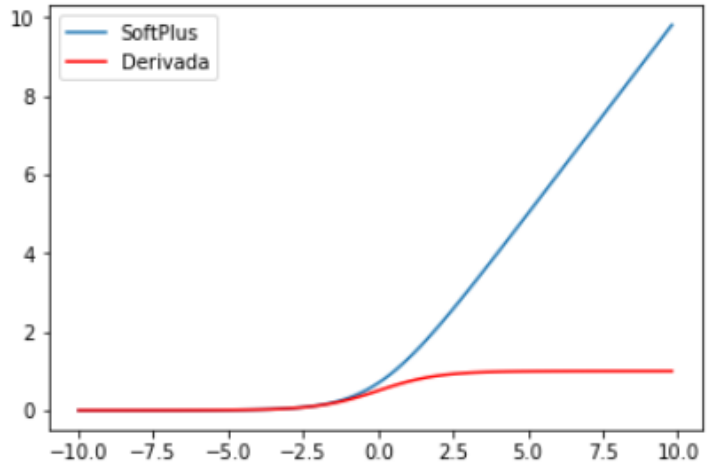


Figura 3.21: Representació d'una SoftPlus.

3.5 Funció de cost

La funció de cost és una part molt important en el disseny de qualsevol xarxa neuronal, ja que definirà el procés d'aprenentatge. Una *loss function* és una funció $f(\cdot)$ que depèn de com a mínim dos paràmetres, el valor real y_{real} i el valor generat $\hat{y}_{gen}$. La funció de cost pot ser qualsevol funció que doni una idea de l'error comès per la xarxa per tal de poder actualitzar els pesos en concordança. A continuació s'exposaran les funcions de cost més habituals i es repassarà en profunditat les funcions de cost en una *Generative Adversarial Neural Network*.

3.5.1 Funcions de cost més comunes

De forma general, es pot parlar de dos tipus comuns de xarxes neuronals. Per una banda, les que implementen una funció que retorna un valor. I, per l'altra, les que retornen una probabilitat. Aquestes últimes acostumen a ser xarxes classificadores. Així doncs, hi ha

també dues funcions de cost molt habituals, una per cada una de les tipologies esmentades anteriorment.

Norma L_p:

La norma L_p és una funció que calcula la distància entre dos valors tal que,

$$L_p(x) = |x|_p = (\sum_{n=1}^{N} |x_i|^p)^{1/p} \qquad on \quad x_i = y_{real} - \hat{y}_{gen} \tag{3.21}$$

Existeixen diferents tipus de norma. Cada una interpreta el concepte de distància d'una forma diferent, però això queda fora la visió del projecte. A efectes pràctics, les dues normes més utilitzades són la L_1 i la L_2. De fet, de la norma L_2 se'n deriven diverses funcions conegudes com, per exemple, la funció de mínims quadrats o MSE. Que, donat un vector, consisteix a calcular la norma L_2 i fer la mitjana dels termes. La norma L_1 també és coneguda com a error absolut.

Cross-entropy

En teoria de la informació, la entropia és un concepte utilitzat per calcular la quantitat d'informació que aporta un valor i es calcula tal que,

$$Entropy = -log(p_i) \qquad on \quad log(\cdot) = ln(\cdot) \tag{3.22}$$

En [8], a la primera part introductòria, es dóna una millor explicació del concepte. Tot i això, observant Eq. (3.22) es pot deduir el concepte. Donada una distribució p, els valors amb major probabilitat tindran una entropia més petita, i viceversa. Això implica, com és lògic, que els valors que no tenen gaire tendència a aparèixer aporten més informació que aquells que són comuns.

Així doncs, la Cross-Entropy és una tècnica molt utilitzada en problemes de classificació que utilitza el concepte d'entropia per calcular si la probabilitat que sigui una classe C és correcte o no.

$$CrossEntropy = -\sum_{i}^{C} l_i \cdot log(p_i) \tag{3.23}$$

on l_i és la classe a la qual hauria de pertànyer. Si la probabilitat p_i és pròxima a 1, és a dir, la xarxa classifica correctament, el valor tornat serà petit, ja que no cal actualitzar gaire la xarxa. Per contra, si la xarxa no classifica correctament, retornarà valors elevats.

Per problemes de classificació binària, entre dues classes, existeix la Binary Cross-Entropy. Sovint també anomenada Sigmoid Cross-Entropy, ja que l'última capa de les xarxes classificadores binàries és una funció sigmoide.

$$BCE = -l \cdot log(p) - (1 - l) \cdot log(1 - p) \tag{3.24}$$

3.5.2 Loss function en una GAN

Com s'ha comentat en la Secció 3, el valor de les GAN es troba en el seu procés d'entrenament. La funció de cost d'una xarxa generativa es basa a trobar el *maximum likelihood* entre dues distribucions. Per fer-ho, es treballa amb el concepte de divergència. La divergència és una mesura de distància entre distribucions que treballa amb la idea d'entropia. Hi ha diferents variants i tipus de divergència, tot i que la més comuna és l'anomenada Kullback Leiber (KL) divergence,

$$D_{KL}(p|q) = \int p(x) \cdot log(\frac{p(x)}{q(x)})dx \tag{3.25}$$

Existeix una certa discussió en l'ús d'aquest mètode, ja que la KL divergence no és simètrica. En canvi, la divergència de Jensen-Shannon sí que ho és,

$$D_{JS}(p|q) = \frac{1}{2}\int (p(x) \cdot log(\frac{2p(x)}{p(x)+q(x)}) + q(x) \cdot log(\frac{2q(x)}{p(x)+q(x)}))dx \tag{3.26}$$

Al principi, es creia que escollir la KL divergence sobre la JS divergence era un factor determinant en el bon funcionament de la xarxa. Actualment, aquesta idea no està tan clara [7]. A més a més, com s'ha comentat a l'apartat 3, han aparegut noves mètriques i tècniques per mesurar l'error en una GAN [26]. En aquest projecte s'exposarà només la funció definida per Ian Goodfellow que, a data de realització del treball, és la idea més acceptada tot i els seus desavantatges.

Deixant de banda les discussions matemàtiques, la funció d'error pel Discriminant és clara. Se sap que la seva missió és categoritzar les dades reals com a reals i les dades generades com a *fake*. Així doncs, amb el concepte de divergència explicat anteriorment i sabent que l'esperança matemàtica d'una distribució es calcula tal que,

$$E[x] = \int_{-\infty}^{\infty} xf(x)dx \tag{3.27}$$

llavors, la funció d'error del Discriminant es calcula com:

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = \frac{1}{2}[E_{x\sim p_{data}} logD(x) + E_z log(1 - D(G(z)))] \tag{3.28}$$

on el primer terme,$E_{x\sim p_{data}} logD(x)$, representa la probabilitat que l'*input* x sigui real, i el segon terme, $E_z log(1 - D(G(z)))$, indica la probabilitat que l'*output* del Generador sigui *fake*.

L'objectiu del procés d'aprenentatge és maximitzar la funció $J^{(D)}(\theta^{(D)}, \theta^{(G)})$, que és equivalent a maximitzar la probabilitat que D encerti, per tal que el Discriminant discrimini bé les dades. Realment, com es pot veure en la Figura 3.22, per termes d'optimització, el que es fa és minimitzar la probabilitat que D s'equivoqui, és a dir, $-J^{(D)}(\theta^{(D)}, \theta^{(G)})$.

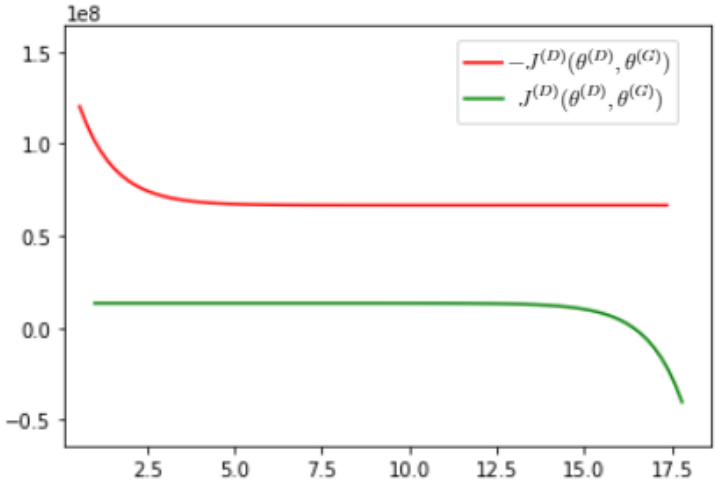


Figura 3.22: S'observa com la funció $-J^{(D)}(\theta^{(D)}, \theta^{(G)})$ presenta una zona amb major gradient a l'inici, el qual és beneficiós en el procés d'entrenament. Font [19].

Definida la funció pel Discriminant, cal fer el propi amb el Generador. Inicialment, es va plantejar que la funció del Generador fos la inversa del Discriminant, $J^{(G)} = -J^{(D)}$. És a dir, l'entrenament del Generador consisteix a intentar enganyar el Discriminant, fer que s'equivoqui. D'aquesta forma, l'entrenament d'una GAN queda plantejat com un *zero-sum game* tal que el procés es converteix en un problema *minimax* [6].

$$C(G, D) = arg\ min_G\ max_D V(D, G) \tag{3.29}$$

Aquest plantejament és correcte en àmbit teòric, però deixa mancances en àmbit pràctic. És per això, que Ian Goodfellow proposa una millora heurística que suposa un joc que disminueix les possibilitats de saturació. Goodfellow proposa mantenir la funció pel Discriminant $J^{(D)}$ i canviar la funció del Generador a

$$J^{(G)} = -\frac{1}{2}E_z[log(D(G(z)))] \tag{3.30}$$

És a dir, G intenta maximitzar la possibilitat que el Discriminant s'equivoqui en discriminar els seus *outputs*. Com s'ha comentat en el cas de $J^{(D)}$, s'observa com la funció del Generador està multiplicada per -1. Això és realment perquè no maximitzem $J^{(G)}$, si no el minimitzem.

3.6 Backpropagation

Un cop calculat l'error, el següent pas és propagar-lo per la xarxa per tal d'actualitzar els paràmetres. Aquest procés es realitza amb la tècnica anomenada *backpropagation*, que no deixa de ser aplicar la regla de la cadena en derivar per tal de trobar el gradient de la funció d'error respecte cada un dels pesos de la xarxa. Per un exemple del funcionament

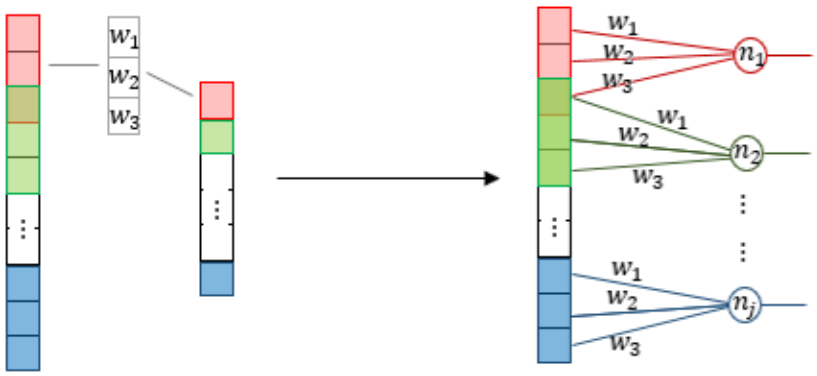


Figura 3.23: Extrapolació d'un procés de convolució a una *feed-forward neural network*.

de la *backpropagation*, veure Annex 1.

3.6.1 Discussió

L'aparició del mètode de *backpropagation* va ser un factor clau en el desenvolupament de les xarxes neuronals. Gràcies a l'ús de gradients per actualitzar les xarxes, aquestes van poder desenvolupar el seu potencial. De fet, calcular el gradient de la *loss function* respecte d'un pes w és equivalent a trobar quin percentatge de l'error correspon a la neurona que alimenta aquell pes.

Tot i això, treballar amb gradients té les seves limitacions i, per aquest motiu, el mètode de propagació endarrere està discutit. Per exemple, per problemes com els de *vanishin gradients* o *exploding gradients* ja comentats en la Secció 2. Investigadors com Yann LeCun són defensors del mètode, tot i que s'està fent recerca per intentar optimitzar la forma en què les xarxes s'actualitzen, com es comentarà a la Secció 7.

3.6.2 Backpropagation en xarxes convolucionals

En una xarxa convolucional la propagació de l'error es fa seguint el mateix concepte que en el cas bàsic (veure Annex 1). Ara bé, cal considerar que en una convolució no es tenen neurones com a tal, sinó que els pesos es troben en els filtres o *kernel*, els quals són vectors, matrius o tensors. Per entendre les relacions internes en una CNN i realitzar la *backpropagation*, es poden extrapol·lar les convulucions a una *feed-forward neural network*, tal com es mostra en la Figura 3.23.

3.6.3 Backpropagation en xarxes recurrents

En una xarxa recurrent la propagació de l'error és un algoritme complex a causa de l'estructura d'aquest tipus de xarxes. Com s'explica en la Secció 2, la informació es propaga d'una cel·la t a la següent $t + 1$. És per això, que l'algoritme usat en RNN

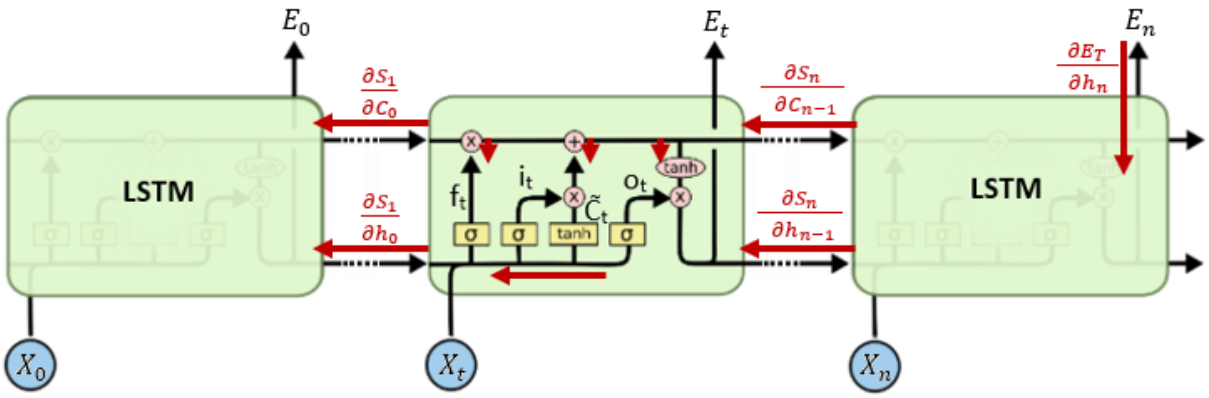


Figura 3.24: Diagrama d'un procés de *Backpropagation Through Time*.

s'anomena Backpropagation Thought Time (BPTT). Com es pot veure en la Figura 3.24, en un procés de BPTT els gradients entren en l'última cel·la i, d'aquesta, es propaguen de cel·la en cel·la fins a la primera.

En una xarxa LSTM, dins de cada cel·la hi ha vuit pesos a optimitzar. Les cel·les es componen de quatre portes f_t, i_t, $\widetilde{C}_t$ i o_t. I, en cada porta, s'utilitzen dos pesos, el que multiplica els *inputs* W_i i el que multiplica el *hidden state* (h_{t-1}) W_h. Per tant, dins la cel·la t el gradient, provinent de la cel·la $t-1$, es propaga per cada porta.

3.6.4 Backpropagation en GANs

En una GAN el procés de càlcul dels gradients es realitza en dues fases. Primer, es generen uns valors que van al Discriminant i, aquest, obté uns resultats i calcula un error. Amb aquest error, es calculen els gradients només dels pesos del Discriminant i s'actualitzen. És a dir, primer només s'entrena el Discriminant. Seguidament, toca entrenar el Generador. Per fer-ho, es realitza el mateix procés que el comentat anteriorment amb la funció d'error de G i els gradients es propaguen fins al Generador. Amb aquesta informació s'actualitzen només els paràmetres de G. Cal comentar que l'ordre d'entrenament pot ser invers. L'únic factor imprescindible és respectar que l'error trobat amb $J^{(D)}$ només es poden actualitzar els pesos de D. Mentre que l'error trobat amb $J^{(G)}$ només és útil per G.

3.6.5 Grafs computacionals

En una implementació real, el mètode de *backpropagation* és molt més senzill. De fet, es resumeix a una simple línia de codi (`.bakward()` en PyTroch). El fet és que si cada vegada que es dissenya una xarxa s'haguessin d'implementar les derivades parcials de cada neurona, el procés seria molt costós, llarg i, inclús, inassumible. És per això, que els *frameworks* i llibreries que s'utilitzen per treballar amb *neural networks* ja incorporen un sistema que ho calcula. El que succeeix és que, en definir una variable, s'emmagatzema informació sobre l'origen i el destí d'aquesta. Així, es crea el que es coneix com a *Com-*

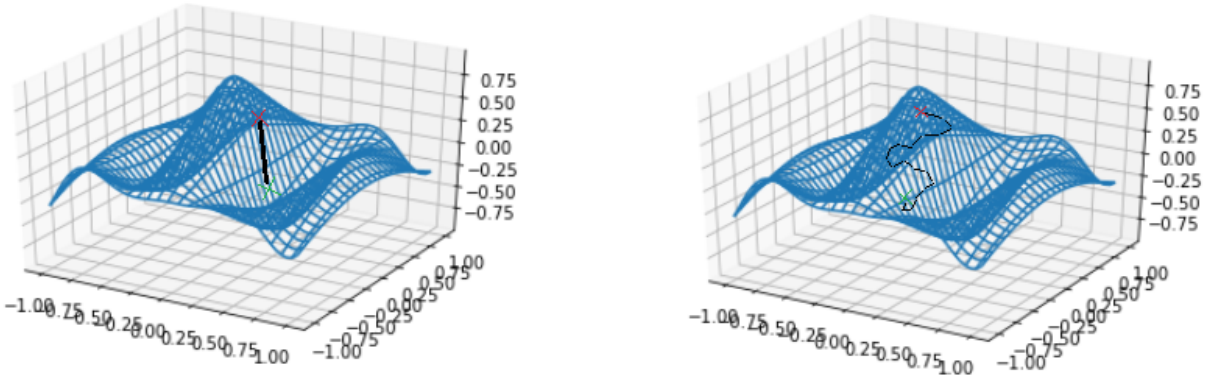


Figura 3.25: (Esquerra) Representació d'un procés de descens pel màxim pendent, d'un punt x_1 (en vermell) a un punt x_2 (en verd). (Dreta) Representació d'un procés SGD.

putational Graph. Llavors, coneixent les relacions internes, automàticament el mòdul pot realitzar la *backpropagation*.

La implementació de grafs computacionals és de gran ajuda pels dissenyadors de xarxes neuronals. Ara bé, alguns docents comenten la importància de no deixar d'entendre com funciona la *backpropagation* de la xarxa dissenyada tot i no haver-hi necessitat real de saber-ho. És el cas d'Andrej Karapahty a Stanford [20].

3.7 Optimització

Com s'ha anat comentant al llarg del treball, el mètode d'actualització dels paràmetres es basa en el càlcul dels gradients i, per aquest motiu, es treballa amb la *backpropagation*. Aquest procés prové del principi amb el qual es planteja l'optimització de la funció implementada en una xarxa neuronal. Concretament, en una xarxa neuronal s'intenta minimitzar la funció de cost i, per fer-ho, es treballa amb el concepte de descens pel màxim pendent o *gradient descent*.

Tal com es pot veure en la Figura 3.25, donada una funció $f(X)$ pertanyent a un espai R^n. Es pot trobar el mínim valor de la funció, X_2, partint d'un punt inicial X_1 si aquest valor es va disminuint de forma iterativa a raó del gradient de la funció avaluada en el punt on es trobi,

$$X_{i+1} = X_i - \eta \cdot \frac{\partial f}{\partial X}(X_i) \tag{3.31}$$

Matemàticament, la derivada de la funció en un punt és el pendent, o inclinació, de la mateixa funció en aquell punt. És per això que la tècnica s'anomena descens pel màxim pendent. η és el que es coneix com a *learning rate* i és un paràmetre utilitzat per estabilitzar el procés d'entrenament. Aquest tipus de paràmetres es coneixen com a *hyperparameters*, perquè són definits prèviament al procés d'entrenament i, durant aquest, no s'actualitzen.

En una xarxa neuronal, aquest mètode resulta realment lent. Cal recordar que es

treballa amb un nombre elevat de pesos i que es requereixen un nombre elevat d'iteracions o *epochs* per assolir resultats òptims en l'entrenament. És per això que de la tècnica de *gradient descent* se'n va derivar l'anomenada *Stochastic Gradient Descent* o SGD. En l'SGD, es realitza el càlcul dels gradients per tots els casos del *training set* obtenint una distribució de gradients per a cada paràmetre. Seguidament, el que és habitual és calcular la mitjana dels valors i, amb ella, actualitzar els pesos. Aquesta tècnica és l'altre extrem del descens pel màxim pendent i, com es pot intuir, és massa dràstica. Per això, en la realitat es treballa amb l'anomenada *mini-batches stochastic gradient descens* que, per comoditat, se segueix anomenant comunament com a SGD. Així que, a partir d'aquest moment, quan es parli d'SGD, s'estarà fent referència a l'ús d'aquesta nova tècnica. En la tècnica de *mini-batches SGD*, s'agrupen les dades del *training set* en subgrups anomenats *batches*. Llavors, per a cada iteració es calcula i s'actualitzen els pesos amb la mitjana dels gradients per a cada *batch*. Per tant, dins cada *epoch*, es realitzen n actualitzacions on n és igual al nombre de *batches* amb el que se separa el *training set*.

El procés d'SGD és més erràtic que la tècnica original de *gradient descent*. És a dir, has de fer més 'passos', reduccions, per arribar a l'òptim. És per això que també és coneguda com a *'drunk man walking'* (veure Figura 3.25). Tot i això, és certament bastant més ràpid i, en conseqüència, eficient.

3.7.1 Discussió

L'SGD segueix sent un model molt utilitzat tot i les variants que han sorgit. De fet, existeix diverses discussions entre quin és el model més apropiat i, en moltes d'elles, l'SGD encara hi surt guanyant. El cert és que cada xarxa és un món i, per tant, no és senzill ni probablement adequat generalitzar un model d'optimització.

Les principals crítiques al model provenen del fet que realment no es coneix com és la funció que es vol minimitzar. L'SGD actualitza el paràmetre θ_t considerant θ_{t-1}, és a dir, és una tècnica amb poca memòria. Aquest factor fa que certs investigadors argumentin que amb l'SGD és més senzill arribar a considerar un mínim local com a global, cosa que seria un error. O, també, asseguren que és un procés certament lent. Per això han sorgit noves tècniques que intenten millorar el procés en aquests aspectes.

En aquest projecte s'ha treballat sobretot en SGD, és per això que la resta de tècniques estudiades en l'actualitat es mostraran en l'Annex 3.

3.8 Hiperparàmetres

Al llarg de les seccions anteriors s'ha pogut observar que en el disseny d'una xarxa neuronal existeixen dos tipus de paràmetres. Els pesos (i, sovint, el biaix) que són l'objecte de modificació durant el procés d'entrenament. I altres paràmetres, com el *learning rate*, que

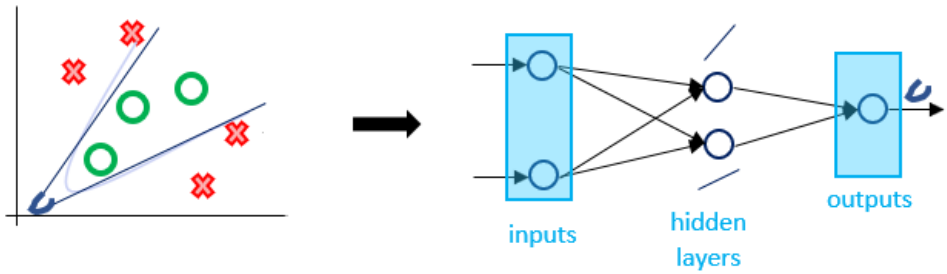


Figura 3.26: Exemple de disseny d'una xarxa neuronal per un problema senzill de classificació.

es fixen al principi de l'entrenament i els quals el seu valor no s'intenta aprendre. Aquests últims són el que es coneix com a *hyperparameters*.

Així doncs, els *hyperparameters* són valors que serveixen per definir la xarxa. Definir-los correctament és molt important per al bon aprenentatge de la xarxa i per assolir una convergència en l'entrenament.

Es poden distingir entre dos tipus d'*hyperparameters*, per una banda els que defineixen l'estructura de la xarxa i, per l'altra, els que defineixen el procés d'aprenentatge. Pel que fa als primers, aquests serien variables com el nombre de capes o el nombre de neurones. Generalment, com més capes té una xarxa, més no-linealitat presenta i, per tant, té major capacitat d'aprendre tasques més complexes. Ara bé, un excés en el nombre de *hidden layers* i/o de neurones pot provocar que la xarxa sigui més complexa que el problema a resoldre i, per tant, que no l'aprengui correctament. Per exemple, en la Figura 3.26 es pot veure un problema senzill del qual se'n pot saber prèviament quin ha de ser el disseny de la xarxa.

El problema és que en una implementació real, habitualment, el sistema requereix major complexitat i no és tan senzill de determinar el nombre de capes i neurones. Segons el professor de la Universitat de Sant Louis a Washington Jeff Heaton [10], el valor òptim de capes es troba entre el nombre d'*inputs* i el nombre d'*outputs*. Tot i això, el rang de valors a provar segueix sent bastant elevat i, generalment, se segueixen dissenyant xarxes amb accés de capes i/o neurones. Ara bé, existeixen tècniques de poda o *pruning* per tal de reduir les dimensions d'una xarxa un cop entrenada. Un exemple en seria el mètode anomenat *Dropout*, que consisteix a eliminar les connexions més febles, amb menys pes, de la xarxa perquè es considera que tenen una menor contribució al resultat final i, per tant, potser sobren. Aquesta, és una tècnica utilitzada pels investigadors de Google i el grau de *dropout* sol ser d'entre un 20 i un 50% del total de connexions. A part de reduir la complexitat de la xarxa i, conseqüentment, augmentar la velocitat d'execució, el procés de poda també protegeix l'algoritme contra l'*overfitting* millorant la capacitat de generalització de la xarxa.

Així doncs, el *dropout* és la tècnica més utilitzada actualment per ajustar el nombre

d'*hyperparameters* lligats a l'estructura de la xarxa un cop dissenyada aquesta. Això no implica que sigui un mètode efectiu. De fet, només és aplicable un cop entrenada la xarxa i coneguts els pesos de les connexions. Recentment, dos investigadors del MIT, Jonathan Frankle i Micheal Carbin, han presentat la hipòtesi del tiquet de loteria [4] on defensen que és més eficient entrenar diversos models de menor dimensió que no pas entrenar un model gran i podar-lo. De fet, els últims estudis sobre xarxes neuronals comenten que la majoria de xarxes que s'estan dissenyant actualment són molt més grans del que realment haurien de ser.

Pel que fa al segon tipus d'*hyperparameters*, aquests serien variables com el nombre d'*epochs* o iteracions, el *batch size*, el *learning rate* o, inclús, la quanitat de poda o *dropout*. Aquestes variables afecten directament a l'entrenament de la xarxa. La forma més comuna de fixar aquests valors és manualment a partir de l'experiència. És a dir, agafant de referència proves anteriors i/o models que funcionin, s'inicialitzen els valors dels *hyperparameters* i, a partir d'aquí, es van modificant seguint un procés de prova i error. Com es pot observar, aquest no és pas el mètode més eficient, és per això que han sorgit diverses tècniques per tal d'afinar aquests paràmetres. Al final, cal entendre que l'objectiu de l'entrenament és reduir una funció de cost. Així doncs, una de les primeres tècniques per fixar els *hyperparameters* és l'anomenat *grid search*. El que es fa és, donat un conjunt de paràmetres a modificar, fixar un conjunt de valors possibles per a cada variable i, seguidament, analitzar la funció de cost en totes les combinacions possibles. Els resultats es poden representar en una malla, d'aquí el nom, d'entre els quals s'elegeix la combinació que ofereixi menor valor en la *loss function*. Aquest mètode és un sistema poc escalable i presenta un problema conegut com a *the curse of dimensionality*, ja que com més gran és el nombre d'*hyperparameters*, més gran és el nombre de combinacions i el temps d'execució augmenta exponencialment. Una altra idea és la cerca aleatòria o *random search* on es realitza un procés similar al *grid search* però elegint aleatòriament les combinacions. També és poc eficaç.

Segurament, el mètode més estès i funcional fins al moment és l'anomenat *Bayesian Optimitzation*. La tècnica utilitza conceptes de probabilitat per aconseguir trobar la millor combinació d'*hyperparameters* de forma eficaç. Com a idea general, l'optimització bayesiana es fonamenta en un procés semblant al d'entrenament dels paràmetres d'una xarxa però pels *hyperparameters*. Per tal de no allargar el procés, es realitzen iteracions en petits *datasets*, fixant els valors dels pesos, per tal de minimitzar una funció de cost. Com que es treballa amb poques dades, el procés d'optimització dels *hyperparameters* es realitza amb tècniques probabilístiques. Com que en aquest projecte no s'ha arribat a plantejar l'ús d'aquest mètode perquè afegeix costos de còmput i temporals al procés, la tècnica serà exposada en l'Annex 4.

Altres tècniques són entrenar una mateixa xarxa amb diferents paràmetres alhora en paral·lel o, també, s'estan dissenyant xarxes neuronals per entrenar xarxes neuronals.

3.9 Mesures d'actuació

Una vegada entrenada la xarxa és necessari avaluar-ne el funcionament amb unes dades diferents de les utilitzades en l'entrenament, el *test set*. Això es deu a dos dels principals problemes que solen presentar aquest tipus de models. Un és que la xarxa no s'hagi entrenat correctament, *underfitting*, sigui perquè no està ben definida o cal afinar algun *hyperparameter*. L'altre és que la xarxa hagi après només a treballar amb les dades del *training set*, *overfitting*, i, per tant, no sigui capaç de generalitzar la seva funció a altres *datasets*.

Per tal d'avaluar els models, es defineixen unes funcions anomenades mesures d'actuació o *performance metrics*. Escollir correctament les funcions amb què s'avaluarà l'actuació d'una xarxa és molt important, ja que existeix el risc de donar per vàlida una xarxa que no funciona bé o, contràriament, rebutjar un bon model. Alguns investigadors opten per adaptar la *loss function*, d'altres opten per mesures estàndard.

En aquest projecte no s'ha arribat a plantejar l'ús de mesures d'actuació, sinó que s'han utilitzat les funcions emprades a [36] i, per tant, no s'aprofundirà més en el tema. Tot i això, sí que s'ha realitzat una cerca de les funcions més comunes i, per qui ho desitgi, s'exposaran en l'Annex 5.

3.10 Tècniques per augmentar l'eficiència

Gran part de la investigació actual es focalitza en l'estudi de noves tècniques per fer més eficients les xarxes neuronals. Com s'ha comentat al llarg el capítol, les xarxes neuronals són un model amb una gran capacitat, ara bé, són costoses d'entrenar, tant en temps com en computació.

Per una banda, s'estan buscant noves formes per millorar la precisió dels resultats. Un exemple en seria la tècnica de *dropout* per aconseguir la generalització del model, tal com s'ha comentat anteriorment. Una altra tècnica bastant innovadora és el *batch normalization* [15]. Aquesta tècnica es basa en l'estandardització dels valors en algun moment de la xarxa i és un mètode freqüent en estadística. El més comú és relativitzar els valors en una distribució normal de mitjana 0 i desviació estàndard 1, tot i que també es pot plantejar en altres rangs. D'aquesta forma es pot utilitzar una mateixa xarxa per treballar en una mateixa operació amb dades de rangs diferents. En el seu origen, Ioffe i Szegedy van plantejar el *batch normalization* com a mètode a implementar entre capes d'una xarxa per tal de reduir un problema conegut com a *internal covariate shift*. És a dir, sovint en capes internes les xarxes poden arribar a treballar amb canvis extrems en el rang de valors que flueixen i això pot causar un mal aprenentatge.

Pas 1:

$$batch - mean : \mu_B = \frac{1}{m} \cdot \sum_{i=1}^{m} x_i$$

$$batch - variance : \sigma_B^2 \frac{1}{m} \cdot \sum_{i=1}^{m} (x_i - \mu_B)^2$$

Pas 2: Normalització

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Pas 3: Ajust (*scale and shift*)

$$y_i = \gamma \cdot \hat{x}_i + \beta$$

Per altra banda, s'està investigant formes d'augmentar la velocitat dels algorismes. Aquesta problemàtica es pot enfrontar des d'un punt de vista matemàtic, des del *software* o des del *hardware*. Matemàticament, la forma d'augmentar la velocitat dels algorismes és aconseguir xarxes menys denses i mètodes més eficaços. Per això s'estan aplicant les tècniques exposades anteriorment.

Pel que fa al *software*, per una banda hi ha el llenguatge. Com es pot veure en l'Annex 2, la majoria de *frameworks* per treballar amb *deep learning* estan implementats en Python. De fet, la majoria d'investigadors que treballen amb dades prefereixen utilitzar Python, ja que és un llenguatge d'alt nivell, intuïtiu i senzill d'utilitzar. Ara bé, realment C++ és un llenguatge més ràpid que Python. La raó és que C++ és de menor nivell, és a dir, està més a prop del llenguatge màquina i, per tant, s'executa amb major velocitat. Tot i això, els *frameworks* de Python estan implementats sobre C++ per poder treballar sobre CPU. Dins el llenguatge, escollir una llibreria o una altra pot augmentar la velocitat de la xarxa. Hi ha molts softwares diferents, alguns estan més pensats per a un tipus de xarxes que d'altres i, per això, depèn de la implementació a realitzar serà més fàcil i ràpida en una biblioteca que en una altra. Ara bé, alguns investigadors asseguren que, generalment, PyTorch és més ràpida que TensorFlow a causa dels protocols que tenen per accedir a la memòria de l'ordinador.

Per últim, en temes de *harware* és on rau gran part de la culpa de la velocitat del model. Inicialment les xarxes s'entrenaven sobre la CPU, coneguda com a processador, de l'ordinador. Aquest dispositiu presentava unes prestacions de velocitat de càlcul molt lentes en treballar d'escalar en escalar. Per això, es va començar a executar el procés d'entrenament sobre la GPU, coneguda com a targeta gràfica, de l'ordinador, on la capacitat de còmput augmenta a rang vectorial. Sobre una GPU la velocitat és acceptable en comparació a la CPU, però en cal més. Per això, el passat 2017 Google va presentar un dispositiu anomenat TPU. Una TPU és capaç de treballar amb tensors i, per tant,

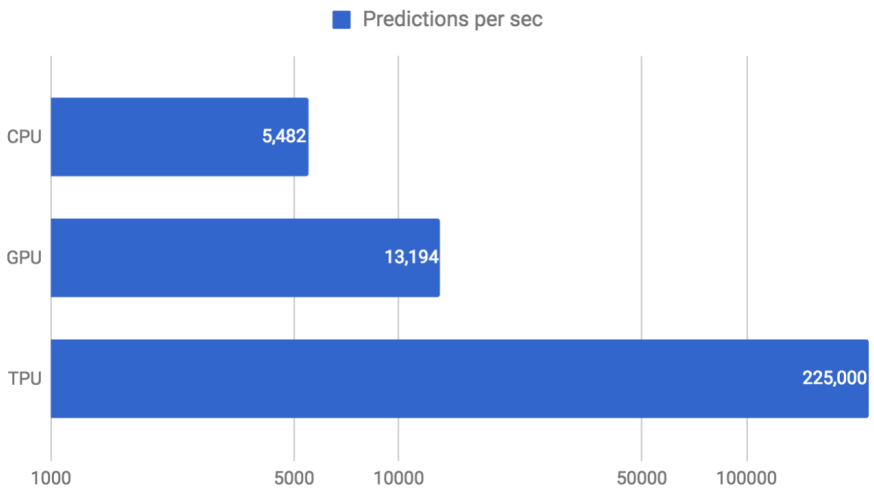


Figura 3.27: Comparativa entre la capacitat d'una CPU, una GPU i una TPU. [27]

augmenta molt la capacitat de càlcul en relació a la GPU (veure Figura 3.27). De moment es pot utilitzar una TPU de Google si es treballa des del núvol, *cloud*, a través de Google Colab. Tot i que no es pot accedir a la màxima potència i algunes estructures complexes com les GAN són difícils d'entrenar. De cara el futur, s'està treballant en la millora d'aquest dispositiu. Segurament, el desenvolupament de sistemes de computació quàntica, de la qual se'n comencen a veure els primers resultats, ajudarà de forma considerable a reduir el temps d'entrenament de les xarxes neuronals.

Paral·lelament a la millora del *hardware*, existeix una branca de computació anomenada *High Performance Computing* que es fonamenta en dissenyar algorismes per fer més eficient l'ús del *hardware*. Així doncs, és molt freqüent realitzar un entrenament distribuït d'una xarxa neuronal. És a dir, fragmentar el procés d'entrenament i repartir cada fragment en una GPU diferent per, posteriorment, tornar a agrupar les parts. Això es coneix com a computació en paral·lel. A trets generals, existeixen dos grans tipus d'entrenament distribuït. Per una banda, l'anomenat *model parallelism*, el qual es fonamenta en separar la xarxa en diferents GPUs. Aquest no és un mètode molt efectiu. Per l'altra, l'anomenat *data parallelism* que consisteix a separar cada *batch* en n paquets més petits i entrenar n xarxes iguals en n GPUs.

Aquest últim tipus és el més utilitzat. Bàsicament es fonamenta en el mateix concepte que l'entrenament per *batches*, és a dir, un cop cada GPU ha calculat els gradients de la xarxa, es calculen uns gradients comuns i s'actualitzen totes les xarxes per igual. En funció de com es planteja aquest procés es pot parlar d'una distribució centralitzada o descentralitzada, si els gradients s'agrupen per entrenar un model comú de referència o no. I síncrona o asíncrona, si totes les xarxes (de cada GPU) s'actualitzen alhora o no. Per més informació, veure [29].

Capítol 4

Definició del Problema

En aquest capítol s'exposarà l'entorn en què es focalitza la implementació de la xarxa, és a dir, el món de la borsa. Es començarà amb una breu introducció de què són els mercats financers que portarà a exposar la inversió en alta freqüència i l'estat actual d'aquesta.

4.1 Mercats financers

Un mercat financer és un espai (físic o virtual) on es comercialitzen (*trade*) o s'intercanvien (*exchange*) valors financers (*securities*) que són actius financers (*assets*), és a dir, parts d'un valor comercialitzat. En altres paraules, és com un mercat tradicional on es compren i es venen productes, amb l'única diferència que, en comptes de peix i verdures, es comercialitzen altres tipus de productes com podria ser una acció d'una empresa.

Per assegurar un bon funcionament dels mercats, existeixen unes institucions que regulen i validen les transaccions de compra-venda. Aquestes institucions són el que es coneix com a borsa. Per exemple, la Borsa de Barcelona o el New York Stock Exchange (NYSE) a Wall Street. També existeixen borses virtuals com la NASDAQ, que regula transaccions en accions a les principals empreses tecnològiques dels Estats Units.

El món financer és complicat per la seva terminologia i complexitat en els productes. Conèixer els diferents mercats i les particularitats dels productes a comercialitzar és important a l'hora d'establir algoritmes d'inversió. Ara bé, aquest treball no es fonamenta en obtenir un bon *portafolio* i un algoritme per operar en ell, sinó en provar el funcionament d'una GAN en dades borsàries. És per això que no s'entrarà en detall de l'estructura del mercat. Tal com es veurà a continuació, al treballar en dades mostrejades cada nanosegon, es desitja operar en un mercat molt volàtil on el preu de l'actiu canviï constantment, és a dir, un mercat on hi hagi moltes transaccions en el mínim temps possible per tal que l'algoritme implementat tingui validesa. És per això que s'ha valorat treballar en mercats de divises com FOREX o Cryptomonedes. Finalment, s'ha optat per treballar amb accions borsàries (*stocks*) d'Apple, tal com es veurà en el Capítol 6.

4.2 Inversió en alta freqüència

Les noves tecnologies han canviat les normes del joc en els mercats financers. La comercialització d'actius financers és una tradició amb llarga història. A l'inici, i fins fa no gaire, era un acte físic amb l'existència d'intermediadors o *brokers*, però amb l'entrada d'Internet, les transaccions van començar a realitzar-se a distància. A més a més, a mesura que els ordinadors guanyaven potència de càlcul, es va començar a operar a través d'ells amb el que es coneix com a *algorithmic trading*. La inversió utilitzant algorismes es basa a implementar tècniques matemàtiques per predir el comportament dels actius i, conseqüentment, comprar-ne o vendre'n. L'ús d'algorismes propicia transaccions a major freqüència i d'aquí en neix el concepte d'inversió d'alta freqüència o *High Frequency Trading* (HFT).

En els seus inicis, l'HFT es realitzava en intervals diaris, però amb l'augment de les capacitats, els intervals s'han reduït a nanosegons. De fet, en 10 anys s'ha passat d'operar de forma horària a processar dades cada nanosegon. És per això que el concepte d'HFT presenta certa divergència en la comunitat inversora, ja que algunes firmes i inversors encara es refereixen a HFT com a invertir en intervals de minuts i no aprofundeixen més. Amb l'augment de la freqüència també ha augmentat l'interès per a aquesta tècnica, de fet el 2000, menys del 10% de les ordres financeres eren en alta freqüència. Mentre que en l'actualitat la xifra representa gairebé el 60% als Estats Units i el 40% a Europa.

Tot i la popularitat de la tècnica, el cert és que el *high frequency trading* té forces detractors, sobretot entre els inversors convencionals. El cert és que l'ús d'algoritmes complica l'anàlisi del mercat, ja que hi ha major nombre de transaccions en poc temps i això augmenta la volatilitat. De fet, un dels successos més rellevants dels últims anys és el *flash crash* del Dow Jones el 6 de maig de 2010, quan per culpa dels algorismes, l'índex borsari es va desplomar i es va recuperar en qüestió de minuts.

4.3 La inversió en l'actualitat

En l'actualitat, segueix essent molt comuna la inversió manual. És a dir, l'anàlisi dels mercats sense utilitzar algorismes. Per una banda, els inversors més experts tendeixen a realitzar anàlisi fonamental on estudien a fons l'estat d'un mercat o valor i realitzen una inversió a mitjà o llarg termini. Per altra banda, i segurament el més comú, és l'anàlisi tècnic. Aquest es basa en l'estudi d'una gràfica de l'evolució d'un valor (veure Figura 4.1) i la cerca de patrons que permetin predir un comportament futur. El gràfic més comú és l'anomenat gràfic de veles on es veu l'evolució a llarg termini d'un valor i el comportament a curt termini en forma de vela, que és la representació en forma de quartils de la volatilitat del preu en un temps determinat.

Figura 4.1: Exemple d'un gràfic de veles. Cada vela representa els quartils de la variació del preu en 1 dia. Una vela vermella indica que en el global del dia el preu ha caigut i una verda indica un augment.

Generalment, la majoria de *softwares* d'inversió o aplicacions treballen amb aquest tipus de tècniques. A més a més, inclouen indicadors per ajudar als inversors. Un indicador és un càlcul, una fórmula, que intenta donar pistes sobre si el preu d'un actiu baixarà o pujarà. Tot i que cal saber interpretar-los, no sempre funcionen correctament i cal saber escollir el moment i el mercat, per tant, l'experiència és un grau. La majoria d'ells, es basen en tècniques simples d'estadística com regressions, mitjanes mòbils, les quals es poden ponderar o no:

$$\hat{y}_{t+1} = \frac{1}{n} \cdot \sum_{i=0}^{n} y_{t-i} \tag{4.1}$$

Aquestes tècniques són poc precises per a ser implementades algorítmicament, és per això que la recerca en *algorithmic trading* es focalitza en models més complexos. S'han intentat implementar diferents tipus de models que engloben idees molt diverses, des d'intentar assimilar un moviment borsari al moviment d'una ona i l'estudi freqüencial d'aquest, a tècniques més estocàstiques basades en moviments aleatoris com és el cas del moviment brownià. També s'ha investigat molt amb conceptes lligats a la teoria del caos. Ara bé, actualment els models financers més utilitzats i amb major ràtio d'encert són els basats en la tècnica ARIMA i en la tècnica GARCH. En termes financers, també se sol anomenar com a model **alpha** a aquell model que obté millors resultats que la resta.

Pel que fa a l'ARIMA (*AutoRegression Integrated Moving Average*), aquesta es basa en la combinació d'un model de regressió i d'un model de mitjana mòbil. Al principi, es treballava combinant directament aquests dos models i per això es parlava de la tècnica ARMA. Ara bé, en els últims temps s'ha realitzat una petita modificació al sistema. Aquesta es basa en no utilitzar una variable directament, sinó diferenciar-la respecte a la sèrie temporal a la qual pertany. Diferenciar una variable vol dir bàsicament no treballar

amb el seu valor en un instant t, sinó utilitzar l'increment del valor en relació a l'instant anterior $t-1$. Realment existeixen diverses maneres de diferenciar una variable, però totes es basen en la idea de relativitzar els valors en una sèrie temporal per tal d'eliminar-ne la tendència i observar l'estacionalitat de les dades. Així doncs, ARIMA es calcula tal que,

$$ARIMA(p,d,q) : Y_t^* = \underbrace{k + \epsilon_t + \sum_{i=1}^{p} \phi_i \cdot Y_t^*}_{AR(p)} + \underbrace{\sum_{i=1}^{q} \theta_i \cdot \epsilon_{t-i}}_{MA(q)} \tag{4.2}$$

on Y_t^* és el valor en t de la variable diferenciada, k és una constant de biaix, ϵ és l'error, ϕ i θ són paràmetres, p i q el rang temporal que es considera per realitzar l'autoregressió (AR) i la mitjana mòbil (MA) respectivament, i d l'ordre de diferenciació que també té a verue amb el rang temporal de valors que s'utilitzen per diferenciar una dada en l'instant t.

Per altra banda, la tècnica GARCH (Generalized AutoRegressive Conditional Heteroskedasticity), té una implementació semblant amb l'ARIMA,

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \beta_i \cdot \sigma_{t-i}^2 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 \tag{4.3}$$

la diferència entre ARIMA i GARCH és que ARIMA treballa amb la idea de mitjana i, per tant, intenta avaluar el comportament de la tendència de les dades, mentre que GARCH treballa amb la variància i, per tant, avalua aspectes lligats a la volatilitat dels valors. És per això que sovint es treballen de forma conjunta.

Aquestes tècniques, tot i que segueixen funcionant, ja tenen una certa edat, i més en el món financer on tot canvia tan ràpidament. És per això que part de la investigació en inversió algorítmica està començant a centrar-se, des de fa un temps, en l'ús de xarxes neuronals, ja que cal trobar nous mètodes de modelar els mercats. Per una banda, perquè les freqüències d'activitat estan augmentant i això els fa sistemes més complexos. Per l'altra, estan apareixent nous mercats com és el cas de les criptomonedes. Fins al moment no es pot parlar d'un gran èxit de l'ús de xarxes neuronals en la borsa. És cert que les grans firmes financeres tampoc publiquen els seus algorismes, ja que no tindrien avantatge a l'hora d'invertir, però el fet és que els diversos *papers* i articles publicats no han assolit tècniques *alpha*.

Capítol 5

Solució inicial

En aquest apartat es farà una breu introducció sobre la recerca en finances utilitzant xarxes neuronals per, seguidament, procedir a exposar les característiques de l'article sobre el qual s'ha iniciat la recerca en aquest projecte.

5.1 Xarxes neuronals en els mercats financers

Tal com s'ha comentat en el Capítol 4, encara no s'han trobat gaires evidències del bon funcionament de xarxes neuronals en dades financeres. Existeixen bastants treballs que han intentat implementar diferents tipus d'estructures per predir el comportament borsari, des de xarxes convolucionals a recurrents passant per simples *feed forward neural networks*. La majoria d'articles parlen d'experiments fallits o bé, en algun cas, d'un funcionament regular que no aconsegueix superar els resultats obtinguts per tècniques com ARIMA-GARCH.

Gran part de la recerca en aquest àmbit es fonamenta en l'estudi del conegut com a *sentimental analysis*. L'anàlisi sentimental consisteix a avaluar les reaccions humanes, tant en premsa com en xarxes socials, per intentar obtenir una correlació entre les expressions humanes i el comportament de certs mercats. Aquest és un àmbit d'estudi ampli que ha assolit certs resultats. De fet, s'ha demostrat que hi ha certa correlació entre algunes fonts d'informació i el comportament de certs valors. Ara bé, encara es troba en fase d'estudi.

Pel que fa a les GAN, l'estructura que és font d'anàlisi en aquest treball, és un sistema sense presència coneguda en l'anàlisi borsari. A data de realització d'aquest treball, només s'ha trobat un article, *Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets* [36], que utilitzi GANs per predir el comportament del mercat. L'article, fruit de la recerca d'un grup d'investigadors xinesos liderats per Xingyu Zhou, va ser publicat el 15 d'abril de 2018 i és el punt de partida d'aquest projecte.

5.2 Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets

En aquest apartat, tal com s'ha comentat, s'exposarà la idea mostrada en [36]. Amb fins explicatius, el model exposat per Xingyu Zhou s'anirà desgranant en diferents subapartats intentant seguir les pautes utilitzades en el Capítol 3.

5.2.1 Dataset

L'article en qüestió avalua la xarxa amb les dades de l'índex xinès CSI 300, en un rang de temps entre l'1 de gener de 2016 i el 31 de desembre de 2016. Concretament, treballa amb dades mostrejades en minuts. Si en el mercat xinès cada període diari d'inversió té 242 intervals en minuts, el nombre total de valors de què es disposa és de 59048. Es pot observar com el treball s'allunya de la freqüència real de l'HFT. Tal com s'ha comentat anteriorment, existeix força confusió entorn el què es considera HFT i el què no. Al Capítol 6 es veurà una discussió sobre les dades utilitzades i sobre quines es treballarà en aquest projecte.

Continuant amb l'anàlisi de les dades, el CSI 300 és un índex que es calcula a partir del comportament de 300 *stocks* comercialitzats en la borsa de Shangai. D'aquests 300 valors, alguns pertanyen a companyies que durant el període d'estudi van sofrir accidents o suspensions, fenomen que genera anomalies en les dades. Per aquest motiu els investigadors van decidir filtrar les dades quedant-se amb un *dataset* final de 42 *stocks*.

Com s'ha comentat en el Capítol 2, una xarxa neuronal obté uns *outputs* a partir d'uns *inputs*. En aquest cas, les variables de sortida són els valors de cada un dels 42 *stocks* en el període estudiat. Com a variables d'entrada, els investigadors de [36] van decidir utilitzar el valor de 13 indicadors. Del Capítol 4, un indicador és una mesura utilitzada per un inversor per ajudar-se a predir el comportament d'un valor. Aquests indicadors són dels més comuns entre els *traders* i, de fet, molts d'ells són considerats *alpha*. En la Taula 5.1, s'anomenen els indicadors, els quals es desglossaran de forma més extensa en el Capítol 6.

Per últim, cal comentar la forma amb què empaqueten les dades per a entrenar la xarxa. Com s'ha comentat en el Capítol 3, els *datasets* en una xarxa neuronal se solen separar en 2, un per l'entrenament i un per testejar el model entrenat. Un dels principals problemes en sèries temporals és determinar quin és el rang de temps adequat de dades a utilitzar. Si es treballa en rangs molt curts, és fàcil perdre informació de tendències a llarg termini. Mentre que si es treballa amb rangs molt llargs, es corre el risc de processar informació no transcendent que actuï com a soroll reduint la precisió del model. En aquest treball s'intenta fer una anàlisi del marc temporal. És per això que s'implementa una tècnica anomenada *rolling segmentation* (veure Figura 5.1). Aquesta tècnica es basa

Indicadors
Preu d'obertura
Preu màxim
Preu mínim
Volum
Turnover
Biaix
Bandes de Bollinger
Directional movement index
Mitjana mòbil exponencial
Stochastic index
Mitjana mòbil
MACD
Relative strength index

Taula 5.1: Taula d'indicadors utilitzats.

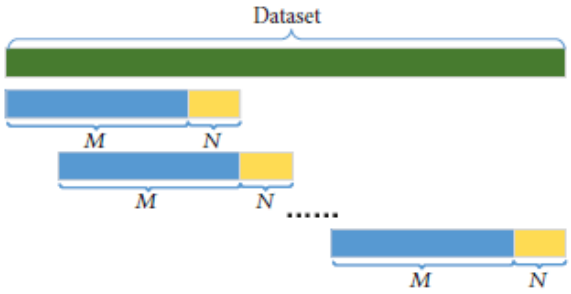


Figura 5.1: Representació del procés de segmentació del *dataset*.

en no treballar amb les 59048 dades de cop, sinó agrupar-les en paquets continus de mida $M + N$. És a dir, l'entrenament d'una xarxa es realitza de forma iterativa entrenant-la diverses vegades amb dades diferents a mesura que es vol predir el comportament en un instant donat. Per tant, primer s'agafa un paquet de M dades com a *training set* i s'avalua la xarxa en un paquet, *test set*, que agafa dades des de l'intant $t = M + 1$ fins a l'instant $t = M + N$. Seguidament, es mou la finestra temporal N intervals tal que com a *training set* s'agafen els valors entre $N + 1$ i $M + N$ i com a *test set* d'$M + N + 1$ fins a $M + 2N$. I així seqüencialment.

En l'article de referència es realitzen diversos experiments per comprovar quin rang temporal és el més adequat, és a dir, fins a quin instant M cal recordar a la xarxa per tal de poder predir amb encert el comportament fins a l'instant $M + N$.

5.2.2 Estructura

La solució plantejada, tal com s'ha comentat, treballa amb una GAN, a la qual anomenen GAN-FD. En tractar-se de dades seqüencials, com a generador s'utilitza una xarxa LSTM

i com a discriminant s'utilitza una xarxa 1-D convolucional. L'estructura de la xarxa pot observar-se en la Figura 5.2.

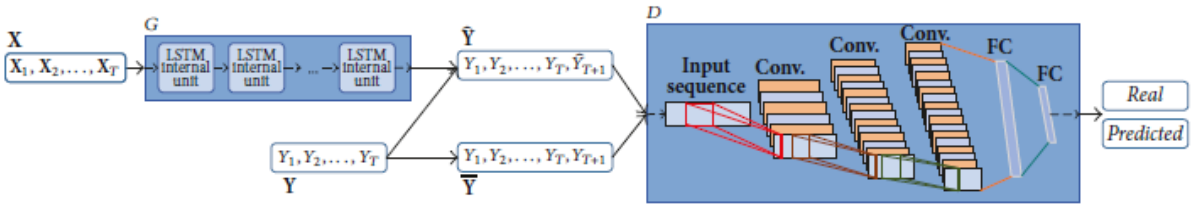


Figura 5.2: Diagrama de l'estructura de la GAN-FD.

Generador

Com a xarxa generadora, s'utilitza una capa de 121 cel·les LSTM, les quals s'han explicat en el Capítol 3. Per entendre el nombre de cel·les, cal entendre la forma en què les dades entren en una cel·la LSTM. Tal com es pot veure en la Figura 5.2, en una xarxa LSTM els *inputs* s'agrupen en grups de 3 dimensions anomenades *batch size*, *time-steps* i *features*.

El *batch size* ja s'ha comentat en el Capítol 3 que era, en aquest cas es treballa amb una dimensió de 121. El nombre de *time-steps* és directament el nombre de cel·les que s'utilitzen, 121. I el nombre de *features* és el nombre de valors que entren per cel·la, en aquest cas 13, el nombre d'indicadors calculats. Per tant, els *inputs* tenen una dimensió de (121,121,13). Aquesta dimensió es deu al nombre d'intervals de minuts que presenta un dia d'inversió en el mercat xinès. És a dir, cada *batch* és l'agrupació de les dades d'un dia. Per aconseguir-ho, es treballa amb un procés semblant al de *rolling segmentation* exposat anteriorment. Per agrupar les dades diàries, el primer *time-step* agafa les dades de l'instant $t = 0$ al $t = 120$ (121 valors), el segon des de $t = 1$ a $t = 121$, i així iterativament fins que l'últim *time-step*, el 121è, recull els valors de $t = 122$ fins a $t = 242$.

Amb aquestes variables d'entrada, la xarxa genera com a *outputs* seqüències de 121 valors per a cada *time-step*. Aquestes seqüències representen la predicció del comportament d'una seqüència des de $\hat{Y}_1$ fins a $\hat{Y}_{T+1}$.

Discriminant

Com a discriminant, els investigadors utilitzen una xarxa convolucional 1-D amb 3 capes de convolució amb un *kernel* de mida 4. Al llarg de les 3 capes, les dimensions de les capes de neurones van creixent, per tant, tècnicament caldria parlar de desconvolució. Al final de cada capa s'aplica una funció d'activació *Leaky* ReLU amb un paràmetre α de 0.01. Com es pot observar de la Taula 5.2, no s'apliquen processos de *pooling* entre capes. Tal com s'ha comentat en el Capítol 3, aquests processos són molt comuns per reduir la dimensionalitat de les dades, sobretot, en tractament d'imatge i en xarxes 2D. En aquest cas no resulta necessari implementar-los, de fet, podrien induir a perdre informació

important. També cal comentar que, després de la segona i la tercera capa de convolució es normalitzen les dades amb un procés de *batch normalitzation* explicat també en el Capítol 3. Per últim, la xarxa realitza un *flattening* després de la tercera convolució i combina les dades en dos *fully connected layers*, l'última dels quals només té dues sortides que passen per una funció sigmoide. Això és deu a què la xarxa discriminant actua com a un classificador que ha de discernir si les seqüències que li arriben com a *input* són reals (1) o fictícies (0).

Capes	Configuració
Convolució 1	Filtre 32x4x1,stride 2,LReLU
Convolució 2	Filtre 64x4x1,stride 2,LReLU,BN
Convolució 3	Filtre 128x4x1,stride 2,LReLU,BN
FC1	128,LReLU
FC2	2,sigmoid

Taula 5.2: Taula de configuració del discriminant.

Pel que fa als *inputs*, el discriminant rep seqüències de 121 valors. Per una banda, rep dades reals, és a dir, la seqüència de valors reals Y_t. Per l'altra, rep una seqüència generada pel generador, la qual hauria de determinar com a falsa, $\hat{Y}_t$. Cal exposar la forma en què es creen aquestes seqüències falses. El fet és que en l'article sota estudi no utilitzen directament els resultats obtinguts en les cel·les LSTM com a *outputs* del generador, sinó que s'agafen només els valors generats per l'última cel·la i es concatenen amb una seqüència de 120 valors reals. És a dir, només es considera el valor referent a l'estat $T + 1$. Aquest fenomen és bastant rellevant en l'entrenament de la xarxa i es discutirà més endavant en el Capítol 7.

5.2.3 Funció de cost

La forma en què s'implementa la funció de cost és també particular. Per una banda, la xarxa s'entrena seguint el típic entrenament antagònic exposat en el Capítol 3. Així doncs, es treballa amb el concepte de *cross-entorpy* tal que el generador intenta maximitzar la probabilitat que el discriminant D classifiqui les dades generades com a reals (1). Mentre que el discriminant intenta maximitzar les probabilitats de classificar les dades generades per G com a falses (0) i les dades reals com a reals (1). Cal recordar que maximitzar les funcions exposades anteriorment és equivalent a minimitzar les mateixes funcions canviades de signe i que, com s'ha vist en el Capítol 3, realitzar aquest canvi pot ajudar en el procés d'entrenament,

$$L^G_{adv}(\hat{Y}) = L_{sce}(D(\hat{Y}, 1)) \tag{5.1}$$

$$L^D_{adv}(Y, \hat{Y}) = L_{sce}(D(Y, 1)) + L_{sce}(D(\hat{Y}, 0)) \tag{5.2}$$

on

$$L_{sce}(A, B) = \sum_i (B_i log(sigmoid(A_i)) + (1 - B_i) log(1 - sigmoid(A_i))) \tag{5.3}$$

Ara bé, des de l'article apunten al fet que minimitzar la *cross-entropy* pot assegurar que el generador enganyi al discriminant, però no que el generador generi seqüències que s'ajustin a la realitat, ja que les dades financeres són complexes. Per això, els investigadors han decidit implementar dues funcions de cost més a la *loss function* del generador. Per una banda, una funció norma L2, vista en el Capítol 3, per considerar la distància entre una seqüència real i una generada,

$$L_2(Y, \hat{Y}) = |Y - \hat{Y}| \tag{5.4}$$

i, per l'altra, una funció dissenyada per considerar l'encert en la direcció del valor. És a dir, realment és més important encertar si un preu pujarà o baixarà, que no pas la quantitat en què ho farà. Encertar el moviment permet reduir el risc en la inversió. És per això que en l'article es dissenya l'anomenada *direction prediction loss*,

$$L_{dpl}(Y, \hat{Y}) = |sign(\hat{Y}_{T+1} - Y_T) - sign(Y_{T+1} - Y_T)| \tag{5.5}$$

Per tant, la funció de cost de G és la suma ponderada de les tres funcions exposades anteriorment. En la implementació de l'algoritme, els investigadors fixen a 1 totes les ponderacions.

$$L^G(Y, \hat{Y}) = \lambda_{adv} \cdot L^G_{adv}(\hat{Y}) + \lambda_2 \cdot L_2(Y, \hat{Y}) + \lambda_{dpl} \cdot L_{dpl}(Y, \hat{Y}) \tag{5.6}$$

5.2.4 Entrenament

Pel que al procés d'entrenament, la xarxa s'entrena seguint els procediments explicats en el Capítol 2 i el Capítol 3. Ja s'ha comentat que es treballa amb un procés de dades segmentades i, per tant, el temps per entrenar un model amb totes les dades augmenta exponencialment.

En relació als hiperparàmetres, els creadors de l'estructura proposen realitzar 10.000 *epochs* amb uns *learning rates* de $\rho_G = 0.0004$ i $\rho_D = 0.02$. Pel que fa als marcs temporals M i N, ja s'ha comentat que des del *paper* es plantegen com dues variables a analitzar i, per tant, es van provant diversos valors al llarg de l'entrenament per tal de definir una dependència temporal correcta.

Amb els *inputs* i els *hyperparameters* definits, el procés d'entrenament segueix el següent algorisme.

Algorithm 1: Procés d'entrenament

Definir ρ_D , ρ_G i la resta d'*hyperparameters*;
Inicialitzar els pesos de la xarxa W_D i W_G seguint una distribució N(0,1);
while *no convergeixi* **do**
    Actualitzar el generador G
    Agafar K noves dades $(X^{(1)}, Y^{(1)})$, ..., $(X^{(K)}, Y^{(K)})$
    $W_G = W_G - \rho_G \sum_t^K \frac{\partial L_G(X^{(t)}, Y^{(t)})}{\partial W_G}$
    Actualitzar el discriminant D
    Agafar K noves dades $(X^{(1)}, Y^{(1)})$, ..., $(X^{(K)}, Y^{(K)})$
    $W_D = W_D - \rho_D \sum_t^K \frac{\partial L_D(X^{(t)}, Y^{(t)})}{\partial W_D}$
end

5.2.5 Resultats

Tal com s'ha comentat anteriorment, en [36] la GAN-FD s'avalua en 42 *stocks* i amb diverses combinacions de longitud de *datasets* M i N. Per tal de calcular la precisió de les dades generades, en l'article es treballa amb dues mesures d'actuació.

Per una banda, es calcula l'RMSRE (veure Annex 5) per tal de tenir una mesura uniforme del grau d'ajust entre les seqüències generades i les reals,

$$RMSRE = \sqrt{\frac{1}{T_0} \cdot \sum_{t=1}^{T_0} (\frac{\hat{Y}_{t+1} - Y_{t+1}}{Y_{t+1}})^2} \tag{5.7}$$

Per l'altra, es calcula el què es coneix com a *Direction Prediction Accuracy* (DPA), una mesura que calcula l'encert en la predicció de la direcció. Així doncs, tal com es fa en les funcions de cost, els investigadors intenten aconseguir un bon càlcul del valor i del sentit, dos aspectes claus en inversió,

$$DPA = \frac{100}{T_0} \cdot \sum_{t=1}^{T_0} I_t \tag{5.8}$$

on I_t és 1 si $(Y_{t+1} - Y_t)(\hat{Y}_{t+1} - Y_t) > 0$, 0 altrament.

Un cop definides les mesures d'avaluació de la GAN-FD, toca testejar-la. Per fer-ho, des de l'article proposen comparar l'actuació de la xarxa amb un model *alpha*, ARIMA-GARCH, un model de *machine learning* tradicional, *Support Vector Machine* SVM, dues xarxes neuronals simples, *feed-forward neural network* ANN i LSTM, i 2 altres GAN amb petites variacions en la *loss function*, GAN-F i GAN-D.

Després d'un llarg procés de comparació, s'observa que la GAN-FD ha aconseguit superar la resta d'algoritmes en la majoria d'*stocks* per la majoria de combinacions d'M i d'N. Concretament, els millors resultats s'han aconseguit amb uns valors (M, N) iguals a $(20, 5)$, on la GAN-FD ha obtingut un millor RMSRE en el 65,08% dels cops i un millor

DPA en 72,24% de les situacions. En la Figura 5.3 es poden observar unes gràfiques representatives dels resultats. L'elevat nombre de valors financers i d'algorismes rivals els fa certament difícils de llegir, però en general, s'observa que la GAN-FD és un mètode *alpha*.

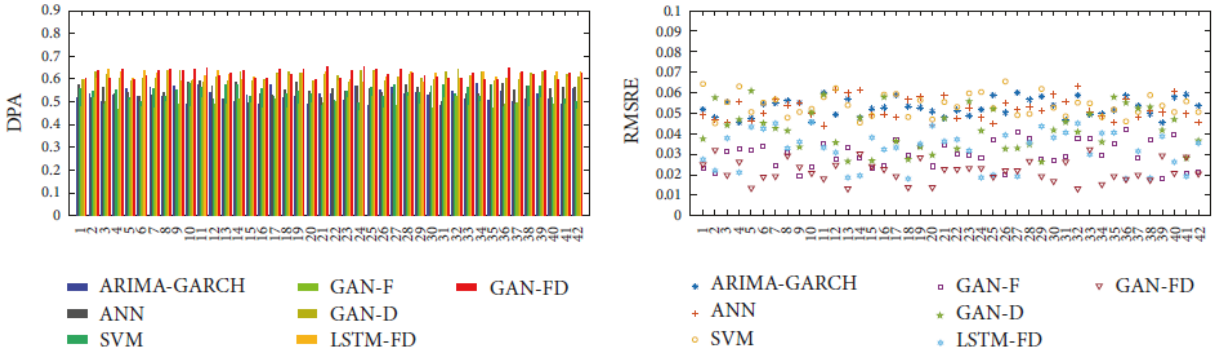


Figura 5.3: Resultats dels diferents algorismes en els 42 *stocks* per $(M, N) = (20, 5)$.

5.2.6 Conclusions

Més enllà de l'anàlisi dels resultats, les conclusions de l'article són més aviat breus. Com a aspectes rellevants, destacar que els investigadors exposen l'avaluació de l'algorisme sota altres condicions, tant temporals com de mercat, com a feina futura.

Capítol 6

Dataset

En aquest capítol es veurà el *dataset* utilitzat per entrenar la GAN. Primer, es començarà exposant les fonts de dades financeres actuals i l'estructura particular de les dades en alta freqüència. Seguidament, s'explicarà com s'han tractat les dades per assimilar-les a les utilitzades en [36]. I, per últim, es realitzarà una discussió temporal sobre les propietats en diferents temps de mostreig.

6.1 Dades financeres

Inicialment, es va intentar obtenir el mateix *dataset* que en [36], és a dir, els valors del CSI 300 durant el 2016. Però no va ser possible, ja que la font proporcionada per l'article no era correcta. Així doncs, es va iniciar un procés de cerca de dades.

Certament, les dades financeres són molt comunes i fàcils de trobar. Existeixen moltes aplicacions i pàgines web que proporcionen les dades, inclús a temps real, de tota mena de mercats. Ara bé, la dificultat d'obtenir unes dades correctes creix exponencialment a mesura que ho fa la freqüència desitjada. És a dir, és senzill trobar dades diàries, però és bastant més difícil torbar dades en segons, per exemple. I encara més si desitges obtenir-les gratis. Cal recordar que per tal de ser el més fidel possible a un problema real d'HFT, en aquest projecte es buscava treballar amb la màxima freqüència possible, preferiblement nanosegons. La dificultat d'obtenir valors a alta freqüència es deu a la dificultat i al cost d'emmagatzemar dades. Guardar valors històrics amb tanta precisió és costós i, generalment, poc útil perquè no hi ha gaire demanda. Les firmes que inverteixen en HFT solen tenir les seves pròpies bases de dades on emmagatzemen les dades que els arriben a temps real. Les grans distribuïdores de dades, com Bloomberg, Morningstar, CQG o els mateixos mercats com NASDAQ, solen oferir amb facilitat dades històriques amb una freqüència màxima de dies i, a partir d'APIs, dades a temps real en minuts. Això es deu al fet que gran part dels inversos realitzen *day trading*, és a dir, inversions diàries i, per tant, amb una informació en minuts en tenen suficient.

Si es desitja major freqüència, cal buscar en aplicacions, fòrums, empreses o institucions especialitzades pels anomenats *quants*. Els *quants* són els inversors que utilitzen models matemàtics per invertir i, per tant, generalment realitzen *algorithmic trading*. En el món dels *quants* les dades en HFT solen anomenar-se *tick-data*. Això es deu al fet que en HFT la microestructura dels mercats és molt més important que a freqüències menors i això produeix que sigui molt més destacable el fet que les dades financeres són discretes. És a dir, no es produeixen transaccions en cada instant i, quan es produeixen, aquestes es realitzen en valors discrets. De fet, en la immensa majoria de sistemes monetaris no té sentit parlar en quantitats inferiors a la centèsima. Per tant, per disminuir espai d'emmagatzematge, en el *tick-data* només es guarden les transaccions i en l'instant en què s'han produït.

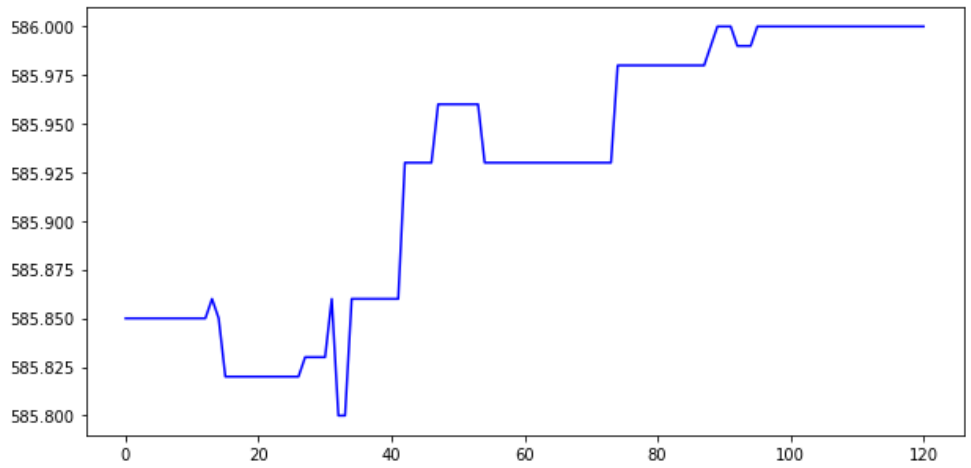


Figura 6.1: Gràfica del comportament de les accions d'Apple en nanosegons.

Per obtenir *tick-data*, realitzant una cerca per Internet, es poden trobar algunes webs que ofereixen paquets gratuïts de dades històriques. Ara bé, generalment són en segons o mil·lisegons. També existeixen algunes aplicacions com QuantDataManager o Tickstory que disposen d'un *software* de prova gratuïta per gestionar *datasets*, però també operen amb les mateixes freqüències de segons o mil·lisegons.

Si es volen obtenir dades a major freqüència, es pot treballar amb APIs. Sobretot des de l'auge de les criptomonedes, ha crescut la inversió algorítmica i, conseqüentment, el nombre d'empreses que ofereixen serveis per descarregar-se dades a temps reals. La gran majoria, i sobretot en HFT, són de pagament, tot i que ofereixen períodes de prova gratuïts on es poden descarregar i guardar les dades que es desitgi. Un exemple en seria Kaiko o Oanda. L'inconvenient d'aquests mètodes és que, en primer lloc, la cartera de productes és limitada tant en dimensió com en temps i que cal saber treballar amb bases de dades i APIs, sovint per JAVA.

Una altra opció és intentar contactar amb empreses, ja que algunes ofereixen ajuda a investigadors. Per exemple Bloomberg col·labora amb Harvard i altres universitats

americanes o Morningstar ho fa amb institucions angleses i australianes. Ara bé, al ser grans empreses és difícil arribar a parlar amb el responsable directe de l'emmagatzematge de dades. A Europa, sobretot al sud, el nombre d'empreses destinades al HFT és molt petit i això complica la feina de buscar convenis cooperatius. Cal destacar l'existència d'una *start-up* anomenada Nexow a Barcelona. Una firma inversora amb poc més de quatre anys de vida que treballa amb inversions algorítmiques i en alta freqüència. Per tal d'obtenir dades, es va contactar amb un dels fundadors el qual es va mostrar en tot moment molt accessible a col·laborar, cedint llicències temporals gratuïtes a APIs per tal d'obtenir dades a temps reals sense tantes restriccions com en les aplicacions explicades anteriorment. També, existeixen pàgines web on es poden executar algorismes al núvol i testejar, *backtesting* en terminologia financera, aquests algorismes amb dades històriques. Tres de les principals empreses en aquest tipus d'aplicacions són Quantopian, QuantConnect i Quantgo. Són realment aplicacions molt útils per a *quants* i disposen d'una àmplia cartera de dades accessibles gratuïtament. Aquestes companyies estan molt obertes a col·laborar en investigacions oferint els seus serveis a canvi d'obtenir els permisos per operar amb els algorismes en cas que siguin realment bons. Un dels problemes que presenten però, és el mateix que en les APIs, cal aprendre a treballar amb les aplicacions i els seus mètodes interns.

Arribats aquí es pot comprovar la dificultat de trobar dades en HFT gratuïtes. L'altra opció és pagar per *datasets*, tot i que els preus solen ser bastant elevats, de l'ordre de 4.000 euros per un petit paquet de dades en nanosegons. Ara bé, existeixen dues companyies que ofereixen dades històriques de prova en nanosegons. Una és Nanotick i l'altra Lobster. La primera ofereix dades de diferents mercats, des de derivats a accions i FOREX. A més a més, inclou algorismes de Python per ajudar a visualitzar les dades. Segurament sigui l'empresa que ofereix dades més completes i precises com a prova. Ara bé, cal dir que des d'abril de 2018 la companyia està inoperativa i, des de març de 2019, no es poden ni descarregar els *datasets* de prova. Pel que fa a la segona, Lobster és una institució que col·labora amb la universitat de Viena, i d'altres d'Alemanya, i que ofereix dades, també molt completes, del comportament de les accions de les principals companyies tecnològiques, com són Apple, Google o Amazon. És possible arribar a descarregar fins a 91998 dades d'un mateix valor.

Analitzada la situació, en aquest projecte s'ha treballat amb dades obtingudes de Lobster, concretament, dades de les accions d'Apple en nanosegons referents al període entre les 9:30:00 i les 10:30:00 del matí del 21 de juny de 2012. Segurament, en termes financers, seria més interessant treballar amb mercats més volàtils, com Bitcoin, ara bé les accions d'Apple són un producte amb suficient volum de transaccions per a funcionar.

6.2 Estructura de les dades

Les dades de Lobster s'organitzen en el que es coneix com a llibre d'ordres o *order book*. La borsa funciona igual que qualsevol altre mercat, és a dir, hi ha persones/entitats en possessió d'un producte (en aquest cas accionistes) i persones/entitats que volen adquirir un producte. Per tant, el mercat es compon d'ofertes per vendre una quantitat n_b d'accions a un preu y_b i, per altra banda, de peticions per comprar n_a accions a un preu y_a. Les ofertes de venda s'anomenen *bid orders* i les peticions de compra *ask orders*. Quan una ordre de compra iguala a una de venda, s'arriba a un acord i, per tant, a una transacció. Sovint, se sol representar el conjunt d'ordres en un instant en una gràfica de profunditat o *order book depth*.

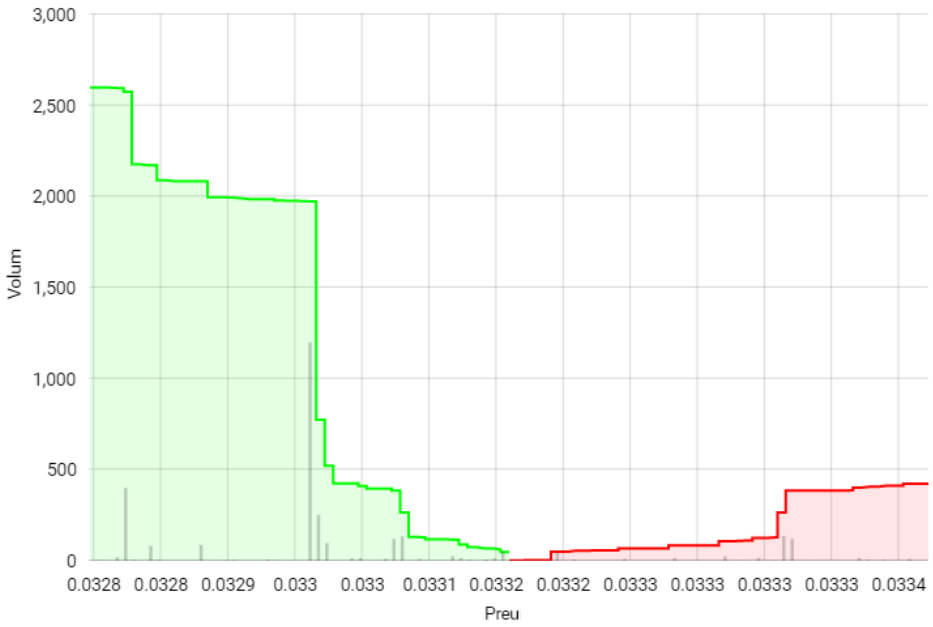


Figura 6.2: Exemple d'una gràfica de profunditat de mercat. En verd, les ordres de compra i en vermell les de venda.

Ara bé, realment els moviments financers són més complexes i, per tant, existeixen diferents tipus d'ordre. Hi ha ordres amb caducitat temporal, d'altres que depèn de l'última transacció realitzada o del valor d'altres productes, n'hi ha que es limiten a comprar una mateixa quantitat d'*stocks* a un mateix preu i d'altres que com a condició restrictiva tenen el preu, entre d'altres. En aquest projecte, es treballarà amb l'evolució del valor de les ordres sense considerar-ne el tipus. Per futures recerques i, amb la intenció d'obtenir un algorisme amb implementació en el món real, seria interessant considerar-ho, ja que afegeix una complexitat en el problema i, alhora, fa un retrat més fidel del problema real.

En les dades de Lobster, s'ofereix tota la informació esmentada anteriorment en dos fitxers separats. Per una banda, en un fitxer s'emmagatzemen totes les ordres rebudes,

tant de compra com de venda, en el període de temps especificat anteriorment. Cada ordre va acompanyada de si és de venda o de compra, el preu, la quantitat o *size*, l'ID de l'operador i del temps en què s'ha produït. Cal esmentar que el temps en HFT s'expressa en UNIX, és a dir, en el nombre de segons que han passat des de mitjanit UTC del dia 1 de gener de 1970. Per l'altra banda, es proporciona un fitxer on, en l'instant de cada nova ordre, s'emmagatzemen les 50 millors ordres de venda i les 50 millors ordres de compra per tal de poder construir el gràfic de profunditat. Com a millors ordres de venda es fa referència a les *bid orders* amb un preu més baix i, per tant, més factibles de què s'arribi a un acord amb elles. I com a millors ordres de compra, a les *ask orders* amb major preu.

6.3 Tractament

Per tal d'imitar la xarxa GAN-FD, s'ha realitzat un tractament de les dades per assimilar-les a les de [36]. Cal comentar que les dades de l'article en qüestió són en minuts, mentre les utilitzades en el treball en nanosegons. Això, en si ja aporta diferència en el seu comportament (a part que són de mercats diferents), cosa que es veurà en el següent apartat. Ara bé, també existeix una diferència en el format. Les dades de l'article estan preparades per treballar en gràfics de veles, (Figura 4.1) o en llibres d'ordres i, per tant, no són grups d'ordres sinó informació sobre les transaccions realitzades en intervals d'un minut. Més concretament, informen sobre la transacció a un preu més elevat i a un preu més petit i el volum total que s'ha comercialitzat.

En aquest projecte, s'intentarà que la GAN modeli el comportament de la major ordre de compra o *ask*. En ser dades de diferent estructura, també canvia la forma en què es calculen els 13 indicadors (Taula 5.1) que són els *inputs* de la xarxa generadora. A continuació, s'exposa com s'ha intentat assimilar el càlcul dels indicadors a partir de les dades disponibles.

Tal com s'ha dit en aquest apartat, cal destacar que les dades en HFT tenen un comportament discret, és a dir, el canvi de preu d'un valor es realitza de forma brusca, no progressiva, entre dos instants consecutius. Això causa que en l'anàlisi instant a instant, hi hagi llargues seqüències amb valors iguals. Alguns investigadors proposen realitzar una interpolació de valors entre els canvis de preu per suavitzar la funció. Ara bé, això també suposa gastar més recursos i temps en aquest canvi i, precisament en alta freqüència, el que no es desitja és gastar més temps del necessari. En aquest projecte es treballarà amb les dades discretes.

Preu d'obertura. El preu d'obertura fa referència al preu del valor a l'inici del dia, en l'obertura dels mercats. És un valor fix que s'utilitza per relativitzar la resta de moviments del valor al llarg de la jornada. En aquest projecte s'utilitzarà el preu de la major ordre de compra a l'inici del període, les 9:30:00.

Preu màxim i preu mínim. El preu màxim i el preu mínim donen una idea de la volatilitat en el preu i sobre el rang de valors que han pres les transaccions en un instant. En aquest projecte s'utilitzarà, per a cada instant de temps, la major *Ask order* i la menor *Bid order* com a preu màxim i mínim. Cal entendre que al final, la propera transacció és molt probable que prengui un dels dos valors, ja que són les ofertes més ben posicionades.

Volum. El volum fa referència a la quantitat d'accions que es comercialitzen, per tant, s'utilitzarà en cada instant la quantitat d'accions de la millor ordre de compra.

Turnover. El *turnover* és la quantitat d'accions comercialitzades respecte al total d'accions que es troben al mercat. És, per tant, una mesura del pes que té una acció en el mercat. Evidentment, una compra d'una gran quantitat d'accions a un cert preu condicionarà molt més les accions futures en el mercat que no pas una compra molt petita. Sabent que el 2012 hi havia 6.629 bilions d'accions d'Apple, llavors

$$Turnover = \frac{Volum_t}{6.629 \cdot 10^9} \tag{6.1}$$

Biaix El biaix és una mesura de tendència que indica si els valors tenen més inclinació a ser superiors a la mitjana o inferiors. Com a mitjana s'entén la mitjana aritmètica mòbil dels T instants anteriors. Així doncs, per calcular el biaix en un instant t, cal primer calcular la mitjana dels últims T valors,

$$\hat{m} = \frac{\sum_{i=t-T}^{t-1} y_i}{T}$$

amb la mitjana calculada cal anar comptant quants valors pertanyen a la part superior de la mitjana, *pos_count*, i quants a la inferior, *neg_count*. Finalment, el biaix es calcula tal que

$$biaix = \frac{pos\_count}{1 + neg\_count} \tag{6.2}$$

Com a T s'utilitzarà 20 intervals, ja que és el més comú en aquest càlcul.

Bandes de Bollinger. Les bandes de Bollinger són una tècnica visual molt usada en anàlisi tècnica. Es coneix que en una distribució normal el 95,4% dels valors es troben en un rang entre $[\hat{m} - 2 \cdot std, \hat{m} + 2 \cdot std]$. Per tant, cal calcular el valor de la mitjana i de la desviació estàndard en els últims T valors,

$$std = \sqrt{\frac{\sum_{i=t-T}^{t-1}(y_i - \hat{m})^2}{T-1}} \tag{6.3}$$

Un cop es tenen els valors d'$\hat{m}$ i de std, les bandes es calculen tal que

$$banda\ superior = \hat{m} + 2 \cdot std$$

$$banda\ mitjana = \hat{m}$$

$$banda\ inferior = \hat{m} - 2 \cdot std \tag{6.4}$$

Com a T s'utilitzarà 20 intervals, ja que és el més comú en aquest càlcul.

Ara bé, en la generador cal entrar-hi un valor i les bandes són massa informació. És per això que s'ha decidit reunir la informació que ofereixen les tres bandes en un sol valor a partir del càlcul del *percent bandwidth*,

$$bandwidth_t = \frac{Preu_t - BandaInferior_t}{BandaSuperior_t - BandaInferior_t} \tag{6.5}$$

Es important veure que en les dades en HFT, en dos instants correlatius de temps, les dades poden no variar, per tant, poden aparèixer errors de valors indeterminats o que tendeixen a infinit. En el *percent bandwidth*, l'únic valor fora del domini és en el cas que $BandaSuperior_t = BandaInferior_t$. En aquesta situació, el que realment succeeix és que no existeixen bandes i, per tant, el valor es troba al mig, com si estigués a la banda central, per tant, $bandwidth = 0.5$.

Directional movement index. El *Directional Movement Index* (DMI) és també una mesura de tendència similar a les bandes de bollinger, però més complexa. De fet, el DMI es calcula amb un indicador anomenat ADX i prèviament al seu càlcul cal realitzar diversos passos intermedis.

En primer lloc cal calcular l'*Average True Range*:

$$TR_t = max[(ask_t - bid_t); (ask_t - ask_{t-1}); (bid_t - ask_{t-1})]$$

$$ATR_0 = \frac{1}{n} \cdot \sum_{t=1}^{n} TR_t$$

$$ATR_t = \frac{ATR_{t-1} \cdot (n-1) + TR_t}{n} \tag{6.6}$$

Com n es sol utilitzar 14 intervals temporals.

Seguidament, es calculen els moviments direccionals. És a dir, es computa si en un interval el valor ha crescut o a disminuït. Per fer-ho, primer es calcula

$$UpMove = ask_t - ask_{t-1}$$

$$DownMove = bid_{t-1} - bid_t \tag{6.7}$$

Llavors, si en un instant t $UpMove > DownMove$ i $UpMove > 0$ es defineix $+DM_t = UpMove$, 0 altrament. Per altra banda, si $DownMove > UpMove$ i $DownMove > 0$ es defineix $-DM_t = DownMove$, 0 altrament.

A continuació, amb els moviments direccionals $(+DM, -DM)$ es calculen els índexs direccionals $(+DI, -DI)$ tal que

$$+DI = 100 \cdot \frac{EMA(+DM)}{ATR}$$

$$-DI = 100 \cdot \frac{EMA(-DM)}{ATR} \tag{6.8}$$

on $EMA(\cdot)$ és la mitjana mòbil exponencial, la qual s'exposarà a continuació, ja que també s'utilitza com a índex.

Finalment, l'ADX es calcula com

$$ADX = \frac{|+DI - (-DI)|}{+DI + (-DI)} \cdot 100 \tag{6.9}$$

Cal comentar que és possible que els moviments direccionals siguin nuls, ja que com s'ha comentat en molts instants consecutius les dades són iguals. En aquest cas, simplement significa que la tendència no varia i, per tant, ADX és 0. El mateix succeeix si ATR és nul.

Mitjana mòbil exponencial. La mitjana mòbil exponencial o EMA també és una mesura de tendència. En ella, es realitza una mitjana mòbil normal i corrent, però els valors es van ponderant amb un paràmetre α.

$$EMA(y) = \frac{1}{T} \cdot (\alpha \cdot y_{t-T} + \alpha(1-\alpha) \cdot y_{t-T-1} + ... + \alpha(1-\alpha)^T \cdot y_t) \tag{6.10}$$

Se sol utilitzar $T = 20$. Pel que fa a α, un valor elevat dóna més pes als valors més recents, mentre que un valor més petit pondera més als valors més antics. En el cas en qüestió, en el càlcul de l'EMA, es treballarà donant més pes als valors recents amb una α de 0.75.

Stochastic index. L'SMI és una mesura del comportament oscil·latori del preu d'un valor. Inicialment es calculen dos paràmetres CM i hl per a cada un dels instant t,

$$CM = ask_{t-1} - \frac{ask_t + bid_t}{2}$$

$$hl = ask_t - bid_t \tag{6.11}$$

sobre els quals es realitzen dos mitjanes mòbils exponencials.

$$cm^{'} = EMA(EMA(CM))$$

$$hl^{'} = EMA(EMA(hl)) \tag{6.12}$$

Llavors es calcula l'SMI tal que

$$SMI = 100 \cdot \frac{cm^{'}}{\frac{hl^{'}}{2}} \tag{6.13}$$

Ara bé, tal i com passava en les bandes de Bollinger, l'SMI és un índex que es sol representar de forma gràfica perquè n'interessa la seva evolució. És per això, que amb la intenció d'aconseguir un valor per introduir a la xarxa, es treballarà amb la senyal:

$$senyal_{SMI} = EMA(SMI) \tag{6.14}$$

Mitjana mòbil. Ja s'ha comentat al llarg de l'apartat. Se sol conèixer com a SMA.

$$SMA = \frac{\sum_{i=t-T}^{t-1} y_i}{T} \tag{6.15}$$

També se sol treballar amb $T = 20$.

MACD. El *Moving Average Convergence Divergence* o MACD també és una mesura seguidora de tendència que es fonamenta en la mitjana mòbil. En aquest cas, el MACD combina EMA de diferents rangs temporals per modelitzar el comportament d'un valor.

$$MACD = EMA(y)_{t=12} - EMA(y)_{t=26} \tag{6.16}$$

Tal i com passa amb l'SMI, en el MACD també es treballa amb la senyal.

$$senyal_{MACD} = EMA(MACD)_{t=9} \tag{6.17}$$

Relative strength index. El RSI és un indicador que mesura l'estat del mercat en funció del pes de certs moviments. És a dir, intenta conèixer si el preu d'un valor està excessivament per sota, o per sobre, el comportament habitual. Un comportament estrany implicaria que s'està sobre-comprant (*overbought*) o sobre-venent (*oversold*) el producte i que, per tant, es probable que hi hagi un retorn a la normalitat en breus.

Per calcular el RSI, primer es calcula si el preu ha crescut ($ask_t - ask_{t-1} > 0$) o, altrament, a disminuït. Si es manté, el RSI és nul. Aquest càlcul es realitza per T instants anteriors a l'instant actual, habitualment, en 14. I es van guardant els moviments en dos llistes separades, els creixents i els decreixent per, a continuació, realitzar una EMA de

les dos. Amb les EMA realitzades, el RSI es calcula tal que,

$$rs = \frac{EMA_{creix}}{EMA_{decreix}} \tag{6.18}$$

$$RSI = 100 - 100 \cdot \frac{1}{1 + rs} \tag{6.19}$$

6.4 Discussió temporal

Tal com s'ha comentat, en aquest projecte es treballarà amb dades en nanosegons, mentre que en l'article [36] es treballa en minuts. Per tal de demostrar les diferències en el comportament en diferents freqüències, en aquest apartat es realitzarà una comparació entre dades d'un mateix valor, accions d'Apple, mostrejades en diferents intervals temporals. Concretament, es treballarà en microsegons, mil·lisegons, segons i minuts. Cal mencionar que no es treballarà en nanosegons perquè Python, en el seu mòdul *DateTime*, no permet precisions de nanosegons a no ser que es treballi en UNIX. Ara bé, a afectes pràctics, treballant en microsegons ja es pot palpar les diferències de comportament.

En primer lloc, tal com ja s'ha comentat prèviament, com exposa el premi Nobel Robert Fry Engle III, les dades són més discretes en freqüències altes que en baixes i això aporta complexitat al seu estudi. Això implica intervals d'observacions irregulars amb zones d'inactivitat. A més a més, tal com es pot veure a la Figura 6.3, en HFT per un mateix interval de temps en minuts, hi ha moltes més dades mostrejades en microsegons que en minuts. Per una banda, aquí rau l'interès de treballar amb tanta freqüència, ja que dóna la possibilitat de realitzar més operacions en menys temps. Ara bé, alhora implica l'existència de molt de soroll en la microestructura i, conseqüentment, una major complexitat de modelar.

És lògic, per tant, veure que en alta freqüència existeixen més oportunitats d'inversió en menys temps. D'aquí l'interès per la tècnica. Si s'analitzen el nombre de variacions del preu del valor (veure Taula 6.1), queda clar l'avantatge teòric d'invertir en HFT.

Mostreig	Mov. Creix.	Mov. decreix.	Major Seq. creix.	Major Seq. decreix.
Microsegons	3921	4811	1	3
Mil·lisegons	3115	3928	3	2
Segons	756	903	5	7
Minuts	30	28	2	3

Taula 6.1: Taula comparativa dels moviments borsaris entre temps de mostreig.

Ara bé, cal veure també que en HFT el model de les dades és més complex. El fet que en major freqüència existeixin major nombre de dades causa certa variació en la normalitat d'aquestes. De la Taula 6.2 se'n pot observar com hi ha certa variació en els

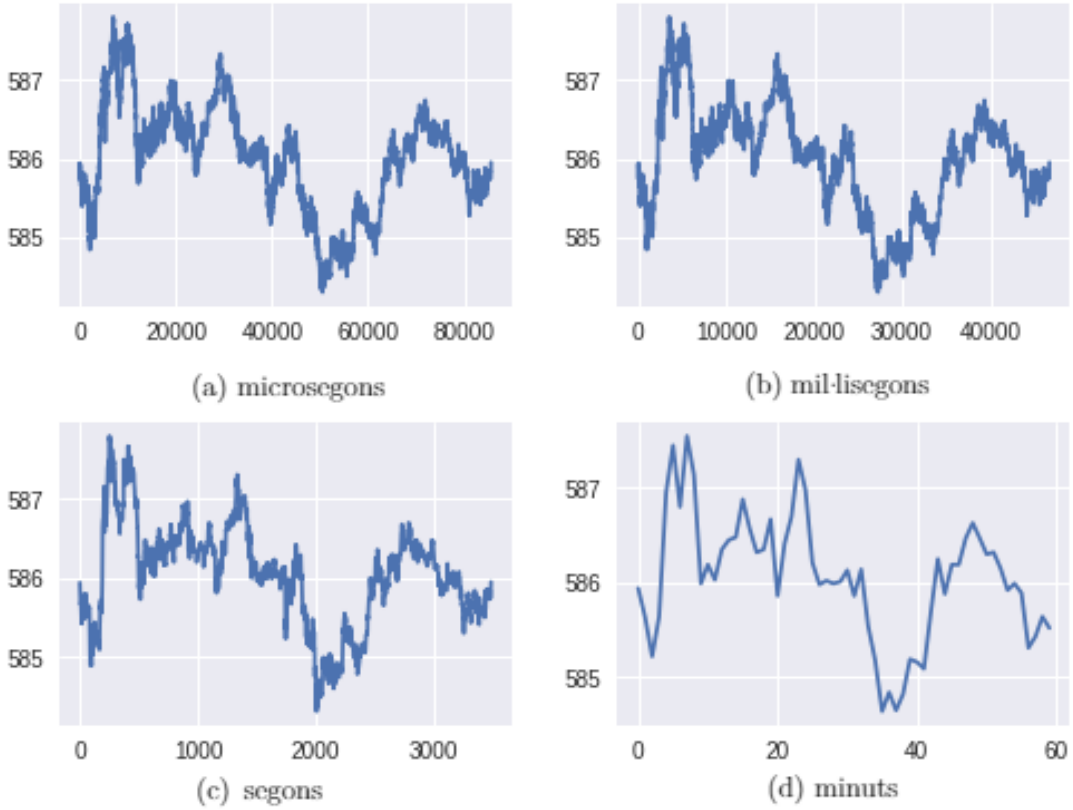


Figura 6.3: Gràfiques del comportament de les accions d'Apple en diferents espais temporals.

diferents mostreigs. Per exemple, en treballar en minuts es perden moltes transaccions i això implica també una reducció en els guanys màxims a assolir.

Mostreig	màxim	mínim	mitjana	std	num. mostres
Microsgons	587.80	584.30	586.0146	0.6831	85594
Mil·lisegons	587.80	584.30	585.9902	0.6885	46737
Segons	587.80	584.31	586.0488	0.6459	3484
Minuts	587.55	584.64	586.0589	0.6609	60

Taula 6.2: Taula comparativa entre temps de mostreig.

Si es representen les dades borsàries, s'observa com segueixen una distribució normal, ara bé, a major freqüència, més irregularitats presenten.

El fet és que les dades en HFT presenten forces irregularitats que fan difícil entendre'n el comportament. Per exemple, solen ser dades asimètriques (o amb *skewness*). També, presenten heterocedasticitat, és a dir, una variació de la variància de les dades en la distribució. En la Figura 6.4, els petits pics que presenta la distribució en la campana. De fet, aquest és el motiu pel qual la tècnica GARCH és tan utilitzada. Per últim, comentar que també solen tenir Curtosis, és a dir, una forta concentració dels valors en un punt. Ja s'ha comentat que en HFT hi ha molts instants on el valor no canvia de preu.

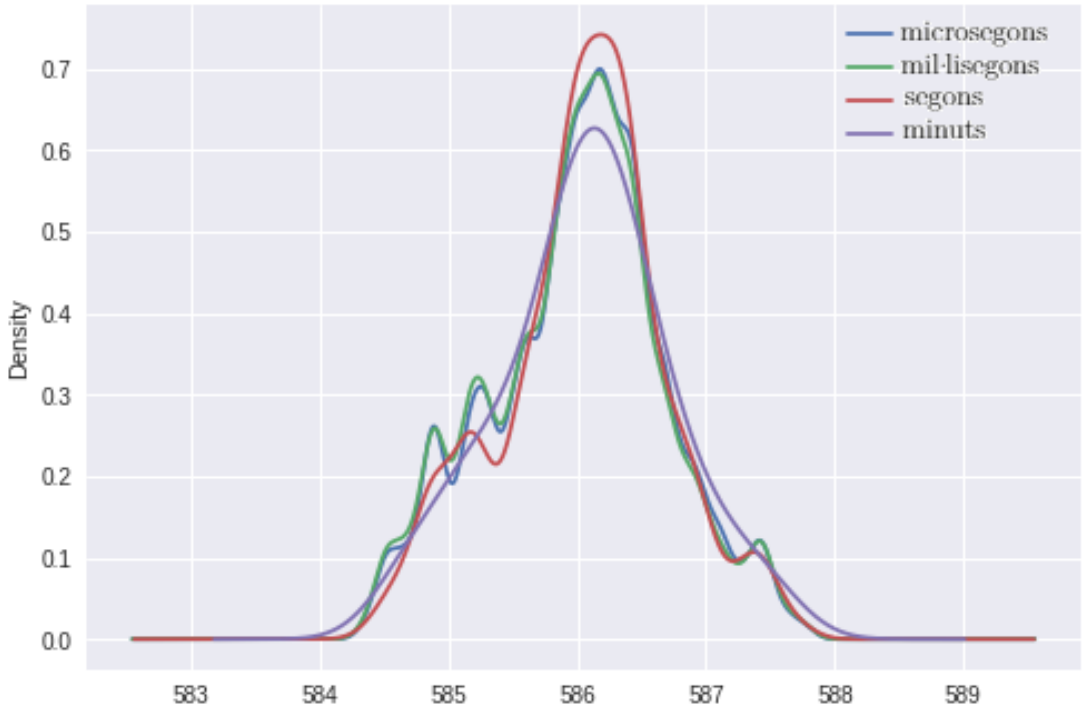
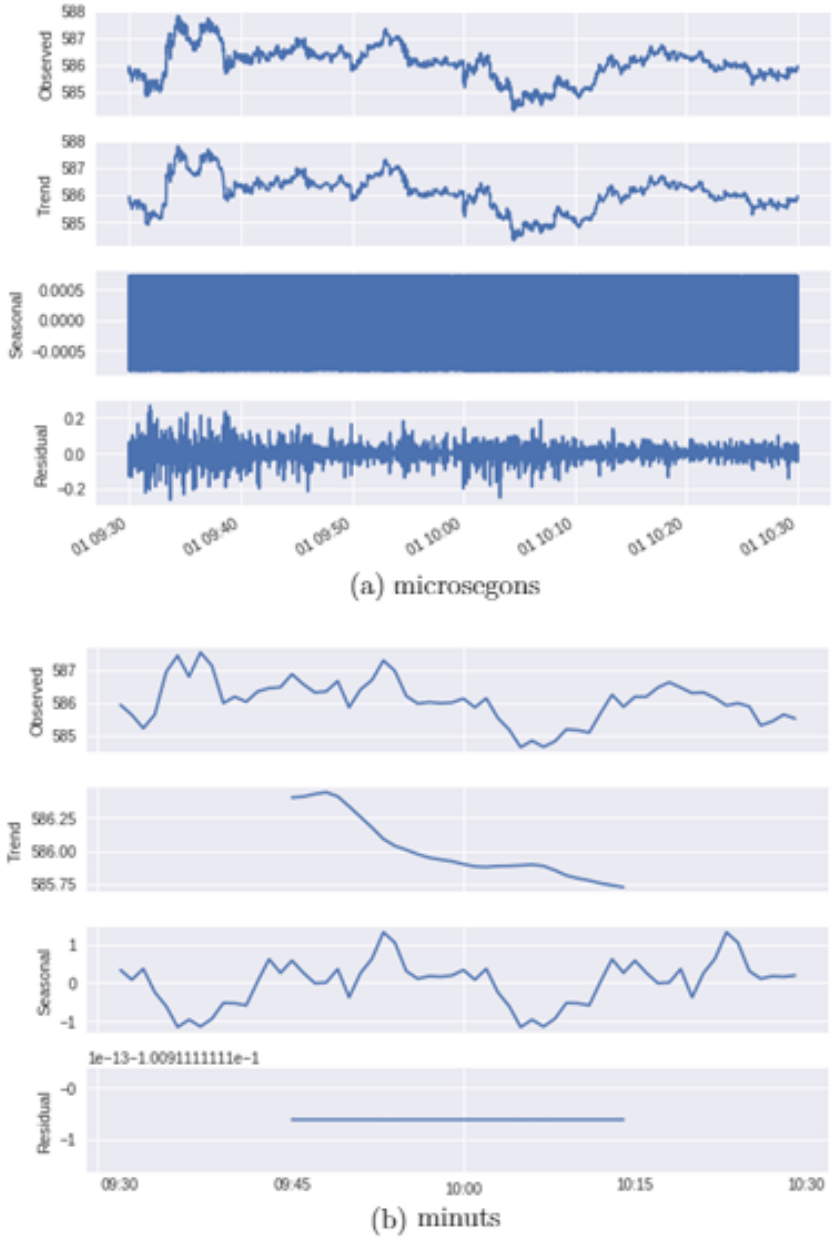


Figura 6.4: Gràfiques de les distribucions de les dades. S'observa com en microsegons (blau) hi ha major irregularitat.

Aquest fenomen va acompanyat de força dispersió entre els valors diferents, per tant, se sol parlar també d'una distribució ampla o amb *fat tail*.

Ja per últim, comentar que en sèries temporals, se sol parlar que les seqüències es poden descompondre en quatre tipus de components. Una referent a la tendència, és a dir, si la mitjana dels valors en un període creix o decreix. Una segona referent a l'estacionalitat de les dades. Sovint, les seqüències van repetint patrons de comportament cada un cert temps. Sovint es confon amb la variació cíclica que reflecteix moviment oscil·latori al llarg del temps. Aquesta component a vegades no es representa. Per últim, un cop s'han extret aquestes tres components de la sèrie general, el que queda es considera soroll. En la Figura 6.5, es pot observar una comparativa entre la descomposició de la seqüència en minuts i en microsegons. Es pot veure com a major freqüència de mostreig, menys comportaments estacionaris s'identifiquen i existeix molt soroll. Aquest soroll és el que fa tan complex l'estudi en HFT.



(a) microsegons

(b) minuts

Figura 6.5: Comparativa de la descomposició de les sèries en minuts i en microsegons.

Capítol 7

Implementació i resultats

En aquest capítol es veuran les diferents implementacions que s'ha realitzat per intentar trobar una xarxa funcional. Inicialment, es parteix del model de [36], explicat en el Capítol 5, i treballant amb les dades explicades en el Capítol 6. Malauradament, aquest sistema va presentar carències en l'execució i va caldre estudiar i realitzar modificacions seguint un procés de prova i error. És per això que en aquest capítol es veuran, per ordre cronològic, els diferents canvis realitzats i el resultat obtingut de la xarxa resultant.

7.1 GAN-FD

Tal com s'ha comentat, el primer pas que es va fer va ser implementar la xarxa neuronal explicada en el Capítol 5. El codi es va implementar en Google Colab i es va decidir treballar en PyTorch. Aquesta decisió va estar sobretot marcada per l'estructura de la xarxa plantejada. En concret, per les funcions de cost. En [36] s'utilitza una funció de cost anomenada *direction prediction loss*, la qual utilitza els dos últims valors d'una seqüència per a computar l'error, Y_{T+1} i Y_T. En un graf estàtic, com el cas de TensorFlow, les variables de la funció de cost són els valors generats, els valors reals i altres variables estàtiques definides abans de l'entrenament i que no varien en ell. En aquest cas, la funció requereix el valor Y_T, el qual és canviant en funció dels *inputs* de la xarxa i, per tant, en cada iteració. Això implica que cal treballar en un graf dinàmic i, per tant, el més aconsellable és utilitzar PyTorch. Pel que fa a les altres dos *loss functions* utilitzades, es poden implementar de forma directa tant en estàtic com en dinàmic i, de fet, tenen comandes ja preestablertes per a treballar-hi. Tot i això, es va decidir implementar de nou totes les funcions utilitzant el mòdul *autograd* de PyTorch.

Resultats. Amb aquest model no es van obtenir bons resultats. Les pèrdues tant del generador com del discriminant no variaven amb el pas de les iteracions. A més a més, el model era extremadament lent d'entrenar. De fet, per realitzar dos *epochs* es tardava un temps aproximat de quatre minuts, i cal recordar que des de l'article original es recomana

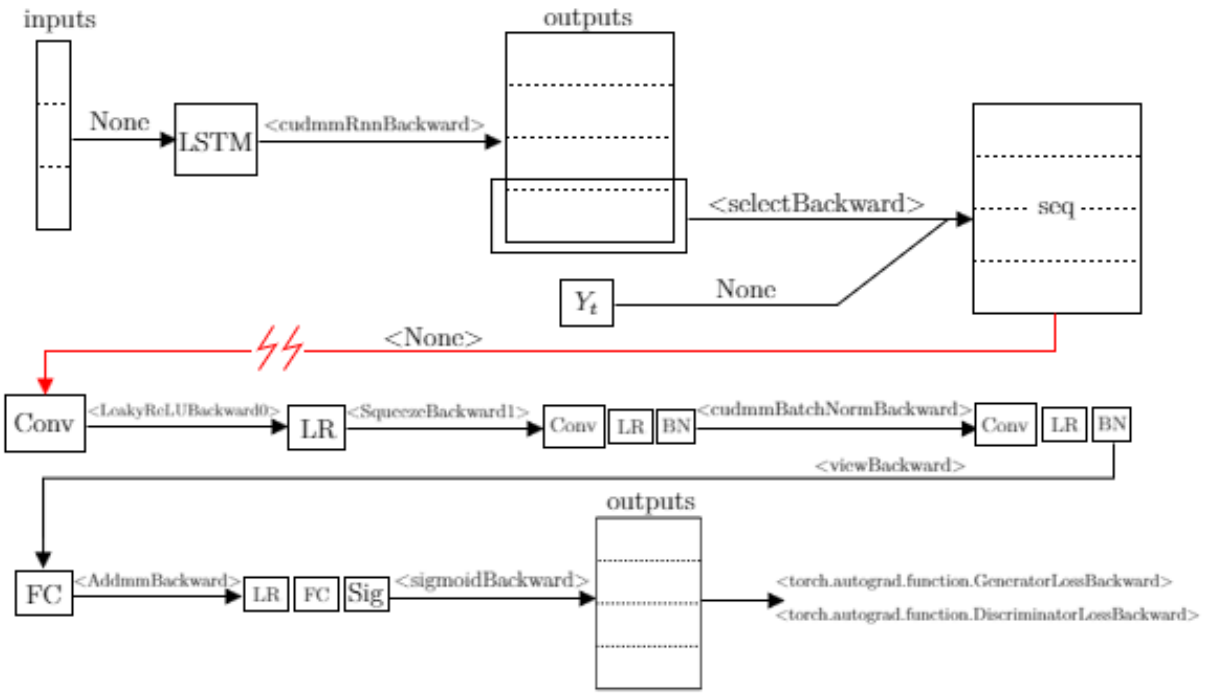


Figura 7.1: Diagrama del graf computacional de la xarxa GAN-FD. S'observa com en el pas del generador al discriminant no es recorda cap relació (<None>).

realitzar 10000 *epochs*. També cal especificar que la potència de còmput oferta per Google Colab va disminuint amb el pas de les iteracions, per tant, encara és més problemàtic que les primeres *epoch* siguin tan lentes.

7.2 GAN-FD_v2

Que una xarxa neuronal no variï l'error en un conjunt d'iteracions no ha de per què ser un error. Hi ha models que tenen una dinàmica lenta i els costa canviar. És per això que es va intentar optimitzar la velocitat d'execució de la GAN-FD. Per fer-ho es van plantejar diverses opcions, exposades en el Capítol 3. Es va veure que el més factible era realitzar un model d'entrenament distribuït i es va decidir treballar amb el mòdul *Data Parallel* de PyTorch.

Resultats. La velocitat d'entrenament canvia de forma exponencial, amb el qual es poden realitzar més iteracions en menys temps. Tot i això, el *loss* de la xarxa seguia sense variar amb el temps i, per tant, es va concloure que existia un error intern d'implementació.

Per observar on estava l'error, es va analitzar capa per capa observant els valors que anava prenent cada neurona. Es va realitzar un procés per assegurar que la funció sigmoide estigués protegida contra valors molt extrems per tal d'evitar que els gradients s'esvaïssin o, per contra, explotessin (*vanishing* o *exploding gradients*). També es va comprovar que la funció de cost estigués ben implementada i realitzés la *backpropagation* correctament.

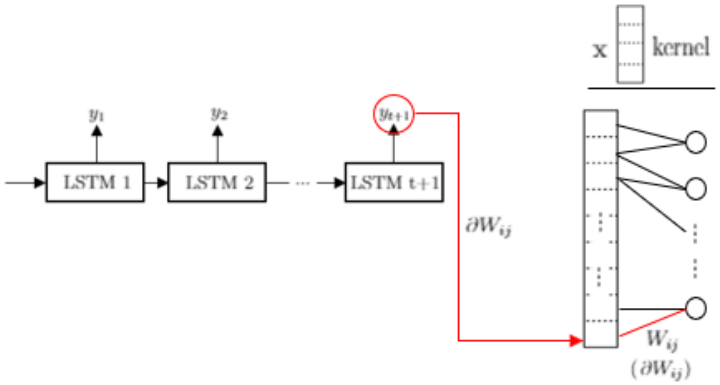


Figura 7.2: Diagrama de la relació de gradients de la xarxa GAN-FD en la unió entre el generador i el discriminant.

Després d'un intens procés de cerca es va observar que hi havia un problema en la connexió entre G i D. El fet és que els *outputs* de G no s'envien directament a D, sinó que D agafa només l'últim valor de la seqüència generada i aquest es concatena a una seqüència real (veure Capítol 5). Aquest procés causava que es trenqués el graf computacional generat per fer el *backward* dels gradients i actualitzar els pesos (veure Figura 7.1).

7.3 GAN-FD_v3

Per tal de no perdre el graf, es va realitzar un nou model amb una nova funció de cost que recollia els gradients del discriminant i els transferia al generador perquè aquest pogués actualitzar-se. Cal recordar l'explicat en el Capítol 3, sobre com tractar una convolució com si fos una *feed forward neural network*. Partint d'aquest coneixement, si s'observa la Figura 7.2, es pot entendre com l'únic valor que cal propagar endarrere és el pes pertanyent a les connexions de l'últim valor dels *inputs* del discriminant que és l'únic valor de la seqüència que prové del generador.

Cal comentar també, que en aquest model es va realitzar una petita modificació en la funció de cost. El fet és que en el generador s'utilitza una funció de cost anomenada *direction prediction loss* que es compon d'una funció signe. La funció signe és una funció peculiar amb una derivada que només pren dos valors, o 0 si x és diferent de 0 o infinit en el 0. Per tant, és una funció que pot portar problemes en entrenar la xarxa. És per això que es va decidir substituir la funció signe per una sigmoide amb un paràmetre $\lambda = 16$, ja que és una funció similar a l'original, però que presenta una derivada més suau.

Resultats. Amb aquesta variació s'obtenia un millor entrenament, ja que s'observava una evolució en el *loss*. Ara bé, al cap de poques iteracions els valors s'estancaven i la xarxa deixava d'actualitzar-se. Per tal de trobar on fallava la xarxa, es va decidir comprovar com fluïen els gradients, també conegut com el *gradient flow* (Figura 7.3 i Figura 7.4).

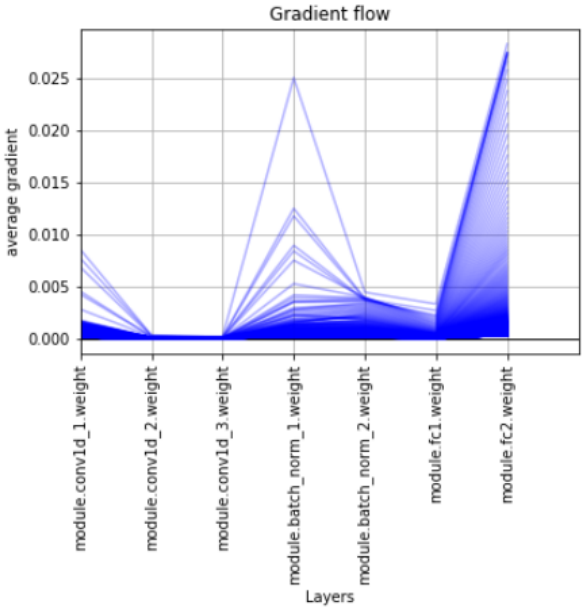


Figura 7.3: Representació del *gradient flow* del discriminant.

Es va comprovar que en el discriminant cada capa rebia gradients correctament i, per tant, s'actualitzaven bé els pesos, mentre que en el generador només hi havia gradients significatius en les últimes capes.

7.4 GAN-FD_v4

Per tal d'augmentar la força dels gradients al generador, es va decidir millorar la connexió entre xarxes. Per fer-ho, es va canviar la forma de tractar els *outputs* de G. Així doncs, en el nou model GAN-FD_v4 els valors generats pel generador es transmetien directament al discriminant. De fet, el model compta amb 121 cel·les que generen una seqüència de 121 valors. Per tant, no és necessari quedar-se només amb l'últim valor general i unir-lo a una seqüència bona tal com es fa en [36]. És més interessant, ràpid i pràctic que el generador aprengui a generar seqüències senceres, i no només un últim valor.

Resultats. En la Figura 7.5, es pot observar com el gradient en el generador augmenta considerablement, ara bé, segueix sense propagar-se més enllà de les últimes 7-5 capes.

Després d'un procés d'investigació, tal com s'ha comentat en el Capítol 3, es va veure que el problema es trobava en un excés del nombre de cel·les LSTM.

7.5 GAN-FD_v5

En l'article [36] no es detalla la programació de la xarxa. Per tant, no se sap quin mètode utilitzen per combatre les dificultats d'aprendre sobre sèries molt llargues en

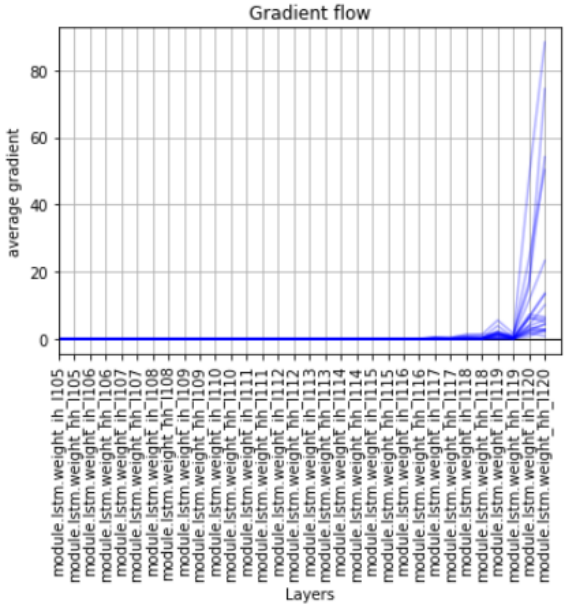


Figura 7.4: Representació del *gradient flow* de les últimes capes del generador.

xarxes LSTM. En aquest projecte, es van plantejar diverses opcions per solucionar el problema. La més factible, còmode i eficient de fer-ho va ser reduir el nombre de cel·les i implementar un sistema de codificació i descodificació, *Encoder-Decoder*, per tal de poder seguir utilitzant les mateixes seqüències com a *inputs*. Pel que fa al nombre de cel·les, inicialment es van plantejar 5 cel·les, ja que és el nombre de capes que els gradients aconseguien traspassar, habitualment, en els models anteriors. Finalment, després de comprovar diverses dimensions, es va optar per treballar amb 3 cel·les.

Resultats. Tal com es pot observar en la Figura 7.7, en el nou model els gradients presenten major fluïdesa que en models anteriors. Ara bé, això no implica pas que el model tingués un funcionament correcte. De fet, el model seguia entrenant de forma errònia. El *loss* del discriminant disminuïa fins a ser pràcticament nul, mentre que el *loss* del generador tenia un comportament creixent i, en molts casos, irregular, tal com es pot observar en la Figura 7.8.

7.6 GAN-FD_v6

Després d'analitzar el model GAN-FD_v5, es va veure que potser el problema en l'entrenament venia de l'extremada diferència entre l'error del generador i el del discriminant. En treballar amb una norma L2, a l'inici, G presenta un error molt gran, ja que la sèrie que genera es troba a molta distància dels valors reals, que es troben entorn el 586. Per contra, la *binary cross-entropy* calcula un error més petit de no gaire més que 1. És a dir, el generador genera molt malament mentre que el discriminant no tant i, per tant, és

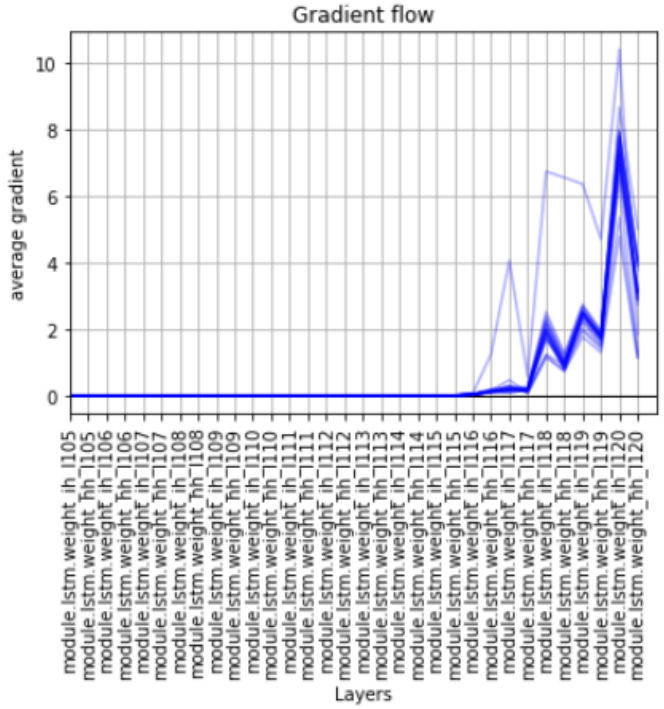


Figura 7.5: Representació del *gradient flow* de les últimes capes del generador.

possible que costi arribar a un equilibri entre ambdós.

És per això, que es va plantejar realitzar un nou model on primer es dediqués un temps a entrenar més G que D, per tal que G generés seqüències més fidels a la realitat en valor. Un cop G ja tenia un error més baix, es tornava a entrenar la xarxa potenciant l'entrenament antagònic perquè G generés seqüències amb una forma semblant a les reals.

Resultats. Inicialment, es va implementar un *learning rate* mil vegades major pel generador que pel discriminant. Concretament, 0.4 i 0.0004 respectivament. D'aquesta forma, l'error de G disminuïa ràpidament alhora que el de D augmentava lentament. Ara bé, un cop s'equilibraven els dos valors, la xarxa deixava d'entrenar. I, per aquest motiu, es va optar per plantejar un canvi en l'optimització un cop el sistema s'equilibrava. Com es pot veure en Figura 7.9, indiferentment de les variacions en els *learning rates*, al canviar les condicions, l'error del generador augmentava i el del discriminant disminuïa.

En vistes d'aquest comportament, es va optar per canviar el mètode d'optimització d'SGD a Adam. Adam té un comportament més agressiu que SGD, tal com es veurà en Figura 7.10. Aquest fenomen fa que, un cop equilibrats els errors, el *loss* de D augmenti de forma considerable i es redueixi el *loss* de G en poques iteracions. Amb aquesta variació, es va assolir un error del discriminant entorn 27.5 i un error en el generador entorn el 0.5. Es va observar però, que amb el temps Adam tendia a desestabilitzar el procés i portar l'error a valors pràcticament nuls. Per aquest motiu es va plantejar un tercer canvi. Un cop aplicat Adam, tornar a SGD per estabilitzar el procés. Certament, el canvi va donar els seus fruits. Ara bé, els errors de la xarxa no variaven. Es va intentar de

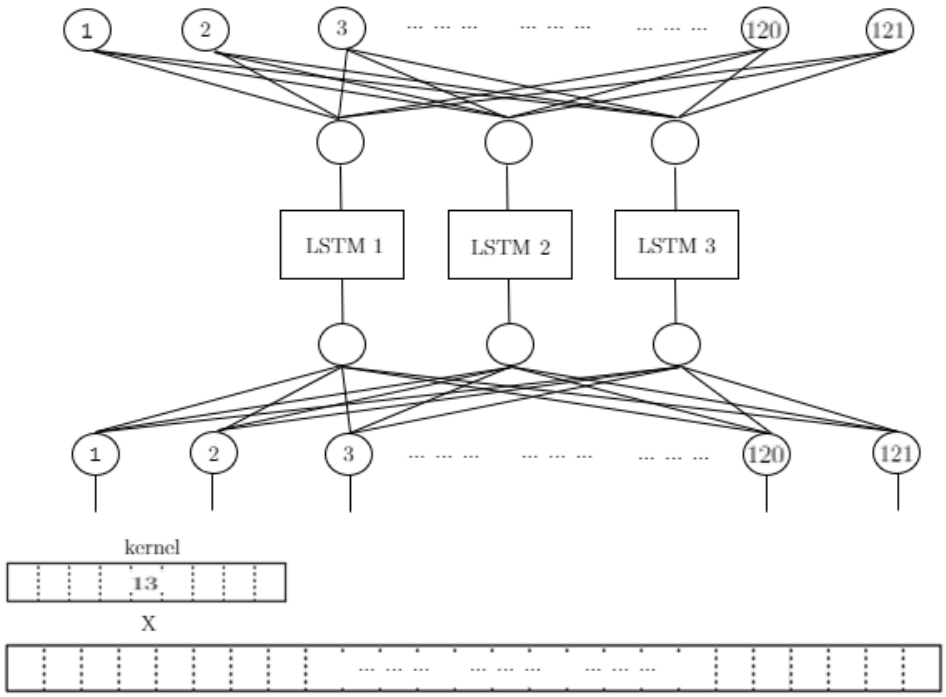


Figura 7.6: Diagrama del nou model de generador.

nou variar els *learning rates*, plantejant un quart canvi, tornant a implementar Adam i, inclús, combinant Adam i SGD un per G i l'altre per D. Tot i això, no es van assolir millors resultats, sempre s'obtenien aproximadament els mateixos valors per molts canvis i *epochs* que es feien. És com si la xarxa arribés a un fals òptim i, per tant, per molt que es variïn les condicions d'entrenament, aquest no varia en considerar que el sistema ja és correcte.

Per tant, finalment, la combinació amb millors resultats va ser:

- De 0 a 250 *epochs* utilitzar SGD amb un *learning rate* de 0.0004 per D i 0.4 per G.
- De 250 a 500 *epochs* utilitzar Adam amb un *learning rate* de 0.004 per D i 0.02 per G.
- A partir de 500 *epochs* utilitzar SGD amb un *learning rate* de 0.004 per D i 0.02 per G.

Si s'analitzen les seqüències generades pel generador després de ser entrenat, com s'observa en Figura 7.11, es pot observar com G genera seqüències amb un valor pròxim al real i amb un comportament força discret. Ara bé, no arriba mai a assemblar-se a la sèrie patró.

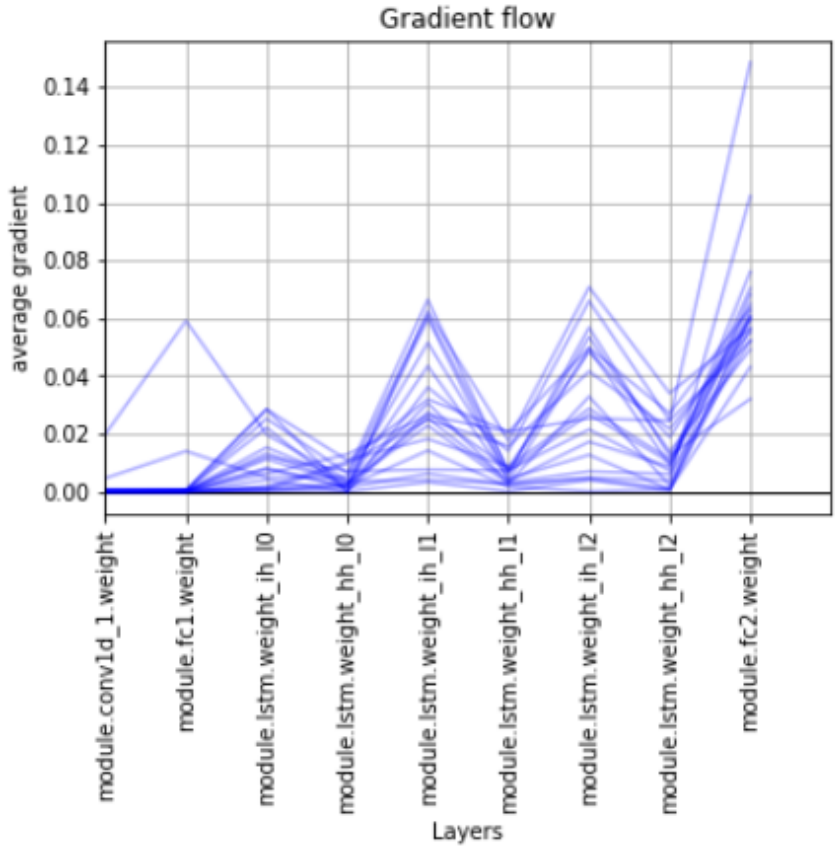


Figura 7.7: Representació del *gradient flow* del nou model de generador reduint el nombre de cel·les LSTM i incorporant un sistema *Encoder-Decoder*.

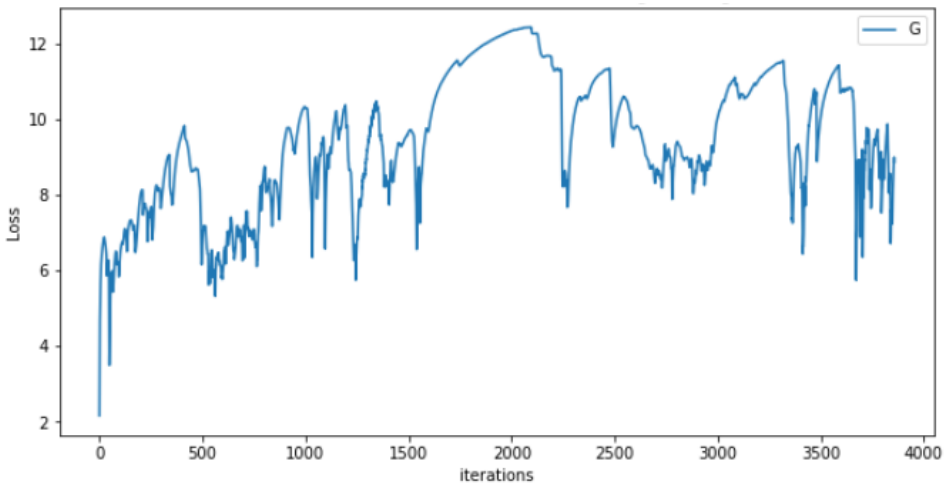


Figura 7.8: Exemple del comportament del *loss* del generador en un procés d'entrenament.

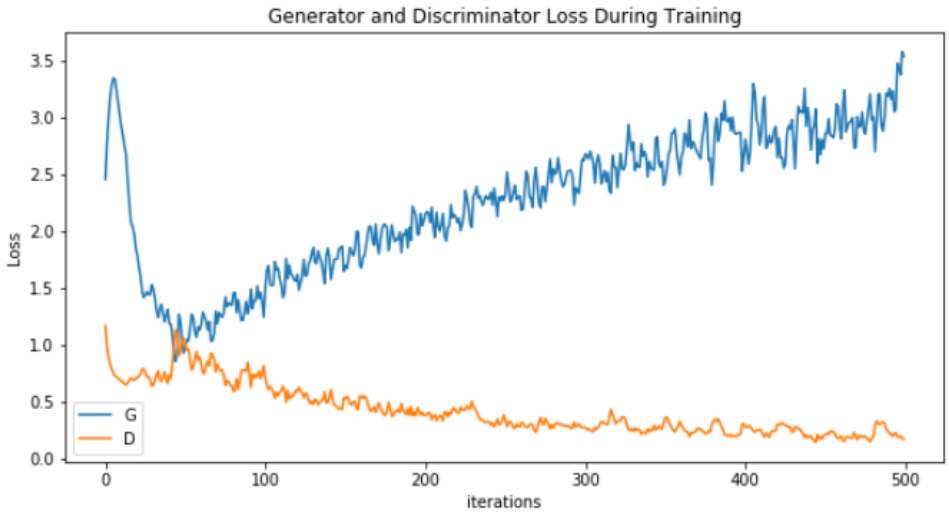


Figura 7.9: Comportament del *loss* al realitzar un entrenament amb un canvi de *learning rate*.

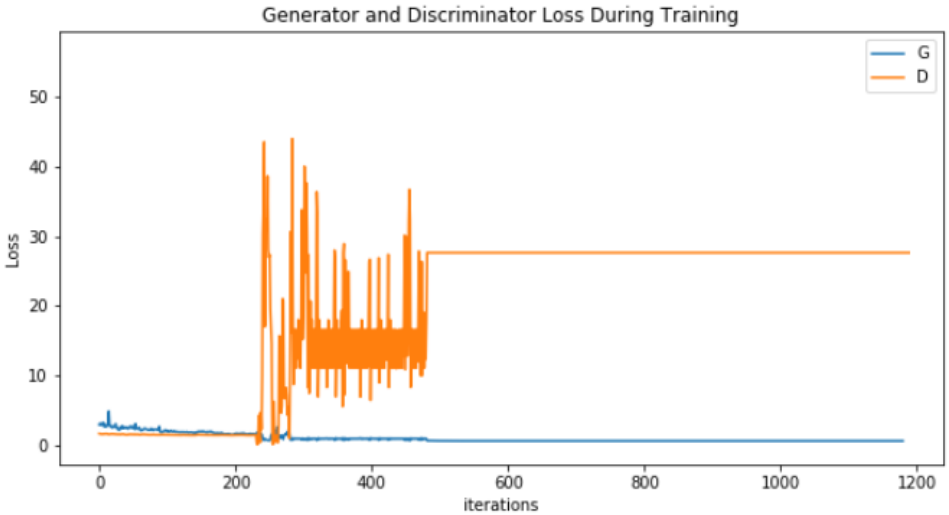


Figura 7.10: Comportament del *loss* en un procés d'entrenament amb 3 canvis en el mètode d'optimització.

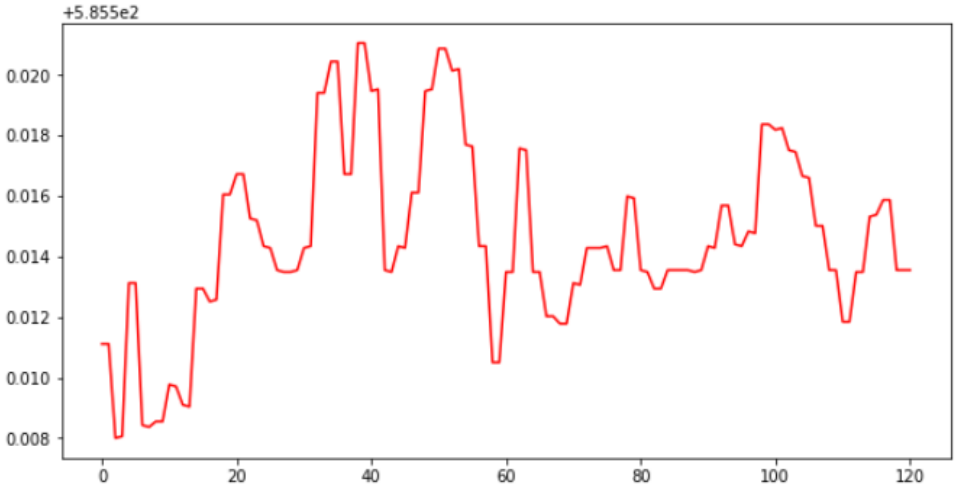


Figura 7.11: Exemple d'una seqüència generada per G després d'entrenar-se.

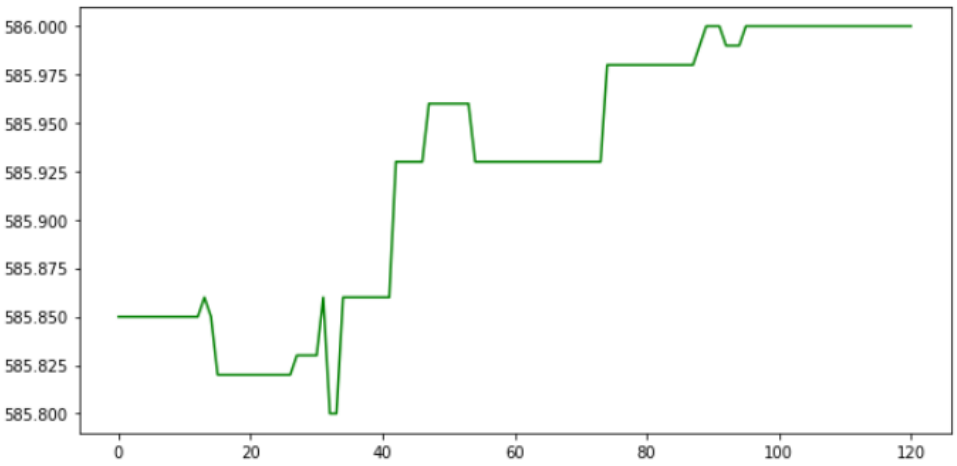


Figura 7.12: Exemple d'una seqüència real.

Conclusions i Treball Futur

Del desenvolupament del projecte se'n poden extreure diverses conclusions lligades als objectius inicials que bàsicament es resumien en investigar la capacitat de les *generative adversarial neural networks* en dades financeres d'alta freqüència.

En primer lloc, en relació a les dades financeres, s'ha pogut observar la dificultat que afegeix l'augment de la freqüència de mostreig en les dades temporals. En el Capítol 6, s'ha vist com les dades en alta freqüència presenten molt soroll i n'és difícil veure la relació entre el senyal i el soroll (*signal-to-noise ratio*). En canvi, en dades mostrejades en major espai temporal es poden distingir tendències i estacionalitats i, per tant, és més senzill implementar models predictius. Alhora, s'ha observat també que treballar en alta freqüència ofereix molts avantatges als inversors, ja que es poden realitzar més operacions en menor temps i, per tant, obtenir més beneficis.

En segon lloc, s'ha comprovat la capacitat de les GAN en aquest tipus de problemàtiques. S'ha observat com és complicat treballar amb aquest tipus de models a causa de la seva inestabilitat a l'hora d'entrenar-los. A més a més, partint del concepte inicial definit en [36], s'ha vist com aquest tipus de xarxes són molt sensibles a canvis en l'entorn i en el *dataset* utilitzat.

Respecte a l'article sobre el qual s'ha iniciat l'estudi hi ha alguns aspectes a discutir. El model emprat és un model certament poc pràctic a causa de l'elevat nombre de cel·les LSTM al generador. Això implica un alt cost d'entrenament i, alhora, un obstacle per la fluïdesa dels gradients. A més a més, la forma en què es comuniquen el generador i el discriminant és poc pràctica. Per una banda, perquè dificulta que el generador aprengui del discriminant, ja que els gradients són molt febles. Per l'altra, perquè el generador en si ja és prou gran com per a aprendre a generar seqüències senceres i no només l'últim valor.

Un altre aspecte a tenir en compte és el plantejament dels *inputs*. Des de l'article es plantegen unes dades en minuts i preparades per a treballar amb veles. Per tant, les dades en nanosegons preparades per treballar amb llibres d'ordres no són equivalents. Això implica que el càlcul dels 13 índexs no té el mateix sentit en un tipus de dades que en d'altres. Cal mencionar que, tal com s'ha comentat en la fase introductòria, un dels avantatges de les xarxes neuronals és el fet de treballar amb *raw data*. Si es calculen els índexs, les dades ja són tractades amb el qual es perd temps i, conseqüentment, eficiència.

Per tant, es pot concloure que el potencial de les GAN en dades borsàries en HFT encara està per explotar. No es posa en dubte que puguin ser un model *alpha* per aquest tipus de problemàtiques, però cal explorar noves estructures per millorar-ne la robustesa, l'estabilitat i l'execució.

A partir d'aquest projecte s'obren nous horitzons com a treball futur. Per una banda, caldria seguir realitzant proves amb el model utilitzat emprant major capacitat de còmput i diferents variacions tant en els hiperparàmetres com en el nombre de capes i neurones de les xarxes.

Per altra banda, i segurament com a millor opció, seria interessant provar noves estructures. S'ha explicat que les GAN són un àrea de recerca molt popular en l'actualitat i, per tant, existeixen molts articles que utilitzen aquests models per treballar en seqüències. Caldria veure quins d'aquests models presenten idees capaces de resoldre la problemàtica en què s'ha treballat. Una de les propostes més interessants és treballar amb tècniques de *reinforcement learning* durant el procés d'entrenament per tal de millorar-lo. En l'article [22] es fa una comparació de diferents mètodes GAN i conclouen que imposar penalitzacions en funció dels gradients ajuda a estabilitzar el procés de convergència. De fet, fa poc que es va presentar un model de xarxa generativa per seqüències anomenada seqGAN [35], la qual treballa amb *reinforcement learning*, i mètodes probabilístics com Monte Carlo, per entrenar la xarxa i augmentar l'estabilitat.

Un altre model interessant seria l'anomenat pathGAN [21]. Una xarxa que precisament utilitza cel·les LSTM al generador i que està pensada per treballar en reconeixement de seqüències en imatges.

Per últim, també seria interessant treballar amb uns *inputs* diferents. Caldria aprofitar les dades que es disposen del llibre d'ordres. En baixa freqüència, els índexs i les tècniques heurístiques per buscar patrons poden ser útils, ja que existeix una major estacionalitat en les dades. En alta freqüència això no succeeix. En HFT la microestructura del mercat té un major pes i, per tant, saber quines són les ordres en un instant t condiciona les transaccions en estats futurs.

Pressupost

El projecte ha estat realitzat al llarg de 20 setmanes, des de la setmana del 4 de febrer de 2019 a la setmana del 17 de juny de 2019. En la Figura 7.13, es pot observar un diagrama de Gantt dels passos seguits en el treball.

De forma resumida, es pot parlar de 3 grans blocs de treball :

- Estudi teòric dels conceptes relacionats amb el projecte. Des del funcionament d'una xarxa neuronal i les seves tipologies (Capítol 3), fins a un estudi conceptual de les sèries temporals i el món financer. En paral·lel, s'ha anat estudiant l'article [36] que ha estat el punt de partida del projecte.

- Cerca, anàlisi i tractament de les dades borsàries en alta freqüència. Explicat en el Capítol 6.

- Implementació de l'estructura GAN-FD [36] en PyTorch i anàlisi dels resultats.

Cal comentar que, inicialment, es preveia comparar l'execució de la GAN-FD amb altres tècniques referents. Ara bé, donat que no s'ha obtingut un funcionament adequat de la xarxa en les dades pertinents, el tercer bloc de treball s'ha estès fins a la finalització del projecte a fi de buscar una estructura funcional.

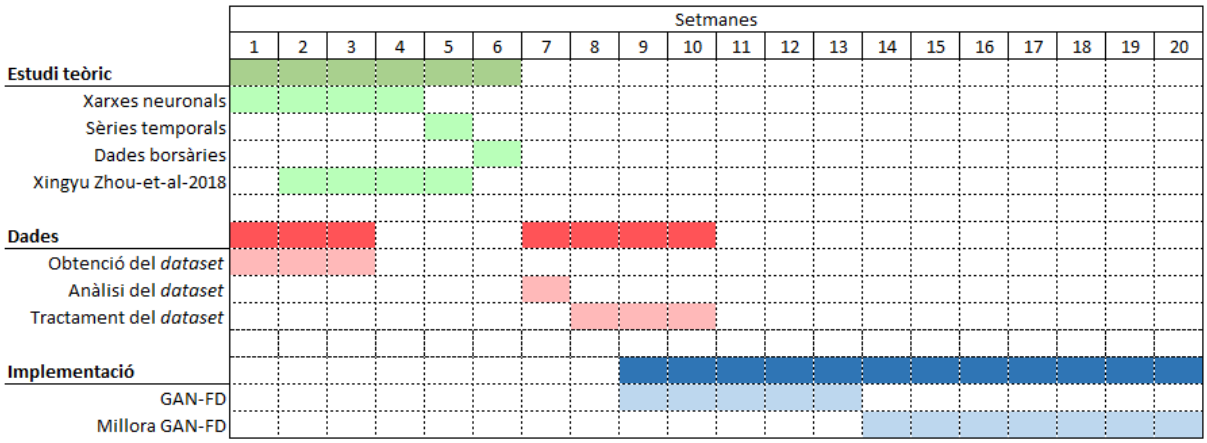


Figura 7.13: Diagrama de Gantt de les 20 setmanes de duració del projecte.

Conegut el repartiment del temps invertit en el projecte, cal considerar-ne els costos. Es calcula una dedicació mitjana de 25 hores a la setmana. A més a més, durant el primer

i l'últim més, hi ha hagut, en paral·lel, hores de dedicació a la redacció de la memòria. Amb tot això, es consideraran tres tipus de feines diferents. Unes feines que denominarem 'tipus 1', de caràcter investigador, les quals tenen un cost de 30 € l'hora. Un exemple en seria la primera fase d'estudi i la fase d'anàlisi tant de les dades com de la GAN-FD. En segon lloc, un 'tipus 2' de feines de caràcter tècnic amb un cost de 20 €, com per exemple el tractament de les dades i la programació de la xarxa en PyTorch. Per últim, les feines 'tipus 3', amb un cost de 10 € l'hora, on s'engloba, bàsicament, la redacció de la memòria. D'altra banda, cal tenir en compte les despeses referents al material utilitzat. En aquest cas, en tot moment s'ha treballat amb un ordinador portàtil amb accés a Internet. Les dades han estat obtingudes de forma gratuïta i, el codi, desenvolupat en Google Colab, una plataforma també gratuïta. Per tant, només hi ha hagut un cost relatiu a Internet i al cost energètic de l'ús de l'ordinador. Es desglossen els costos seguint la Taula 7.1. En resum, es pot parlar d'un cost aproximat del projecte de 13893 €.

Concepte	Dedicació	Cost horari (€/h)	Cost total (€)
Tasques tipus 1	350 h	30	10500
Tasques tipus 2	150 h	20	3000
Tasques tipus 3	24 h	10	240
Electricitat	524 h	0.0058	3*
Internet	5 mesos	30*	150
Cost total			13893

Taula 7.1: Taula de costos. *S'ha considerat una despesa energètica horària de 0.40 kWh, un preu de 0.115 € el kWh i un cost d'Internet mensual.

Impacte ambiental

En relació a l'impacte ambiental, el projecte en qüestió no té un efecte directe al medi ambient. El treball està destinat al món financer i utilitza tècniques matemàtiques i informàtiques per a realitzar-se. Evidentment, tot procés comporta unes despeses o residus. En aquest cas, com a impacte ambiental l'únic que es pot considerar és la despesa elèctrica de l'ordinador amb què s'ha treballat. De la Taula 7.1 se sap que s'han consumit aproximadament 210 kWh d'energia. Tenint en compte un valor de mix elèctric peninsular de 321 g CO_2/kWh, la petjada de carboni associada al projecte en CO_2 equivalent al consum elèctric, el resultat es tradueix a aproximadament 67,4 kg CO_2.

En el projecte realitzat, per tant, l'impacte ambiental no és gaire elevat. Ara bé, el món del *machine learning* i el món borsari sí que tenen un impacte energètic important [31]. Són dos àmbits on es treballa amb diversos ordinadors, pantalles i processadors durant molta estona tant per resoldre complexos algorismes com per estar alerta dels moviments de diversos productes financers. Això dispara el consum energètic i, per tant, afecta el medi ambient. A més a més, es requereixen màquines amb alta capacitat de processament, i encara més en HFT, cosa que implica una renovació periòdica dels ordinadors en favor de nous models més ràpids i eficaços. Això redueix la vida útil d'aquestes màquines que no tenen un reciclatge fàcil. Cal dir que, és cert que els ordinadors poden reutilitzar-se, ara bé, amb la potència a què se sol treballar en aquests sectors, els ordinadors solen sobreescalfar-se sovint i treballar al màxim de capacitat cosa que els desgasta més ràpidament. És per això que, tant per temes mediambientals com per temes de costos, cal tenir compte amb l'ús de tècniques que requereixin alts esforços de còmput i cal investigar per reduir el desgast que generen a les màquines.

Bibliografia

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *Courant Institute of Mathematical Sciences*, 2017.

[2] Jason Brownlee. How to handle very long sequences with long short-term memory recurrent neural networks, Juny 2017. https://machinelearningmastery.com/handle-long-sequences-long-short-term-memory-recurrent-neural-networks/.

[3] Kyunghyun Cho, Junyoung Chung, Caglar Gulcehre, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *Cornell University*, 2014.

[4] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *MIT*, 2019.

[5] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Bioloical cybernetics*, 1980.

[6] Ali Ghodsi. Deep learning, generative adversarial network, Octubre 2017. https://www.youtube.com/watch?v=7G4_Y5rsvi8&t=3159s.

[7] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *NIPS*, 2017.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Cornell University*, 2014.

[10] Jeff Heaton. T81 558:applications of deep neural networks, 2019. https://github.com/jeffheaton/t81_558_deep_learning.

[11] Geoffrey Hinton. What is wrong with convolutional neural nets?, Setembre 2017. https://www.youtube.com/watch?v=Jv1VDdI4vy4.

[12] Geoffrey Hinton, Sara Sabour, and Nicholas Frosst. Dynamic routing between capsules. *Google Brain*, 2017.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Google Brain*, 1997.

[14] Nathan Hubens. Deep inside: Autoencoders, Febrer 2018. https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Google*, 2015.

[16] Akshay Chandra Lagandula. McCulloch-Pitts neuron - mankind's first mathematical model of a biological neuron, July 2018. https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1.

[17] Balaji Lakshminarayanan. Understanding generative adversarial networks, 2018. http://www.gatsby.ucl.ac.uk/ balaji/Understanding-GANs.pdf.

[18] Yann LeCunn, León Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *IEEE*, 1998.

[19] Fei-Fei Li, Justin Johnson, and Serena Yeung. Generative models, Maig 2017. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf.

[20] Fei-Fei Li, Justin Johnson, and Serena Yeung. Convolutional neural networks for visual recognition, Primavera 2017. http://cs231n.stanford.edu/.

[21] Kevin McGuinness Noel E. O'Connor Marc Assens, Xavier Giro-i-Nieto. Pathgan: Visual scanpath prediction with generative adversarial networks. 2018.

[22] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? *Cornell University*, 2018.

[23] Christopher Olah. Understanding lstm networks, Agost 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[24] Max Pechyonkin. Understanding hinton's capsule networks. part i: Intuition, Novembre 2017. https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b.

[25] Alec Radford and Luke Metz. Unsupervised representation learning with deep convolutional generative adversarial networks. *ICLR*, 2016.

[26] Bharath Raj. Advances in generative adversarial networks (gans), Gener 2019. https://medium.com/beyondminds/advances-in-generative-adversarial-networks-7bad57028032.

[27] Kaz Sato, Cliff Young, and David Patterson. An in-depth look at google's first tensor processing unit (tpu), Maig 2017. https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu.

[28] Schinruk. Understanding keras lstms, Agost 2016. https://stackoverflow.com/questions/38714959/understanding-keras-lstms.

[29] Skymind. Distributed deep learning, part 1: An introduction to distributed training of neural networks, Novembre 2017. https://blog.skymind.ai/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks/.

[30] Matthew Stewart. Introduction to turing learning and gans, Maig 2019. https://towardsdatascience.com/comprehensive-introduction-to-turing-learning-and-gans-part-1-81f6d02e644d.

[31] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *MIT*, 2019.

[32] Avinash Sharma V. Understanding activation functions in neural networks, March 2017. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[33] Hanh Vu, Huyn-Chul Kim, and Jong-Hwan Lee. 3d convolutional neural network for feature extraction and classification of sensorimotor fmri data, Juny 2018. https://ww5.aievolution.com/hbm1801/index.cfm?do=abs.viewAbs&abs=3093.

[34] Chi-Feng Wang. The vanishing gradient problem, January 2019. https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

[35] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *University College London*, 2017.

[36] Xingyu Zhou, Zhisong Pan, Guyu Hu, Siqi Tang, and Cheng Zhao. Stock market prediction on high-frequency data using generative adversarial nets. *Hindawi*, 2018.